Appears in the *Proceedings of the Third Symposium on Operating Systems Design and Implementation, New Orleans, USA, February 1999*

Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov
Laboratory for Computer Science,
Massachusetts Institute of Technology,
545 Technology Square, Cambridge, MA 02139
{castro,liskov}@lcs.mit.edu

Abstract

This paper describes a new replication algorithm that is able to tolerate Byzantine faults. We believe that Byzantine-fault-tolerant algorithms will be increasingly important in the future because malicious attacks and software errors are increasingly common and can cause faulty nodes to exhibit arbitrary behavior. Whereas previous algorithms assumed a synchronous system or were too slow to be used in practice, the algorithm described in this paper is practical: it works in asynchronous environments like the Internet and incorporates several important optimizations that improve the response time of previous algorithms by more than an order of magnitude. We implemented a Byzantine-fault-tolerant NFS service using our algorithm and measured its performance. The results show that our service is only 3% slower than a standard unreplicated NFS.

1 Introduction

Malicious attacks and software errors are increasingly common. The growing reliance of industry and government on online information services makes malicious attacks more attractive and makes the consequences of successful attacks more serious. In addition, the number of software errors is increasing due to the growth in size and complexity of software. Since malicious attacks and software errors can cause faulty nodes to exhibit Byzantine (i.e., arbitrary) behavior, Byzantine-fault-tolerant algorithms are increasingly important.

This paper presents a new, *practical* algorithm for state machine replication [17, 34] that tolerates Byzantine faults. The algorithm offers both liveness and safety provided at most $\lfloor \frac{n-1}{3} \rfloor$ out of a total of n replicas are simultaneously faulty. This means that clients eventually receive replies to their requests and those replies are correct according to linearizability [14, 4]. The algorithm works in asynchronous systems like the Internet and it incorporates important optimizations that enable it to perform efficiently.

There is a significant body of work on agreement and replication techniques that tolerate Byzantine faults (starting with [19]). However, most earlier work (e.g., [3, 24, 10]) either concerns techniques designed to demonstrate theoretical feasibility that are too inefficient to be used in practice, or assumes synchrony, i.e., relies on known bounds on message delays and process speeds. The systems closest to ours, Rampart [30] and SecureRing [16], were designed to be practical, but they rely on the synchrony assumption for correctness, which is dangerous in the presence of malicious attacks. An attacker may compromise the safety of a service by delaying non-faulty nodes or the communication between them until they are tagged as faulty and excluded from the replica group. Such a denial-of-service attack is generally easier than gaining control over a non-faulty node.

Our algorithm is not vulnerable to this type of attack because it does not rely on synchrony for safety. In addition, it improves the performance of Rampart and SecureRing by more than an order of magnitude as explained in Section 7. It uses only one message round trip to execute read-only operations and two to execute read-write operations. Also, it uses an efficient authentication scheme based on message authentication codes during normal operation; public-key cryptography, which was cited as the major latency [29] and throughput [22] bottleneck in Rampart, is used only when there are faults.

To evaluate our approach, we implemented a replication library and used it to implement a real service: a Byzantine-fault-tolerant distributed file system that supports the NFS protocol. We used the Andrew benchmark [15] to evaluate the performance of our system. The results show that our system is only 3% slower than the standard NFS daemon in the Digital Unix kernel during normal-case operation.

Thus, the paper makes the following contributions:

- It describes the first state-machine replication protocol that correctly survives Byzantine faults in asynchronous networks.
- It describes a number of important optimizations that allow the algorithm to perform well so that it can be used in real systems.

This research was supported in part by DARPA under contract DABT63-95-C-005, monitored by Army Fort Huachuca, and under contract F30602-98-1-0237, monitored by the Air Force Research Laboratory, and in part by NEC. Miguel Castro was partially supported by a PRAXIS XXI fellowship.

- It describes the implementation of a Byzantine-fault-tolerant distributed file system.
- It provides experimental results that quantify the cost of the replication technique.

The remainder of the paper is organized as follows. We begin by describing our system model, including our failure assumptions. Section 3 describes the problem solved by the algorithm and states correctness conditions. The algorithm is described in Section 4 and some important optimizations are described in Section 5. Section 6 describes our replication library and how we used it to implement a Byzantine-fault-tolerant NFS. Section 7 presents the results of our experiments. Section 8 discusses related work. We conclude with a summary of what we have accomplished and a discussion of future research directions.

2 System Model

We assume an asynchronous distributed system where nodes are connected by a network. The network may fail to deliver messages, delay them, duplicate them, or deliver them out of order.

We use a Byzantine failure model, i.e., faulty nodes may behave arbitrarily, subject only to the restriction mentioned below. We assume independent node failures. For this assumption to be true in the presence of malicious attacks, some steps need to be taken, e.g., each node should run different implementations of the service code and operating system and should have a different root password and a different administrator. It is possible to obtain different implementations from the same code base [28] and for low degrees of replication one can buy operating systems from different vendors. N-version programming, i.e., different teams of programmers produce different implementations, is another option for some services.

We use cryptographic techniques to prevent spoofing and replays and to detect corrupted messages. Our messages contain public-key signatures [33], message authentication codes [36], and message digests produced by collision-resistant hash functions [32]. We denote a message m signed by node i as $\langle m \rangle_{\sigma_i}$ and the digest of message m by $D(m)$. We follow the common practice of signing a digest of a message and appending it to the plaintext of the message rather than signing the full message ($\langle m \rangle_{\sigma_i}$ should be interpreted in this way). All replicas know the others' public keys to verify signatures.

We allow for a very strong adversary that can coordinate faulty nodes, delay communication, or delay correct nodes in order to cause the most damage to the replicated service. We do assume that the adversary cannot delay correct nodes indefinitely. We also assume that the adversary (and the faulty nodes it controls) are computationally bound so that (with very high probability) it is unable to subvert the cryptographic techniques mentioned above. For example, the adversary cannot produce a valid signature of a non-faulty node, compute the information summarized by a digest from the digest, or find two messages with the same digest. The cryptographic techniques we use are thought to have these properties [33, 36, 32].

3 Service Properties

Our algorithm can be used to implement any deterministic replicated *service* with a *state* and some *operations*. The operations are not restricted to simple reads or writes of portions of the service state; they can perform arbitrary deterministic computations using the state and operation arguments. Clients issue requests to the replicated service to invoke operations and block waiting for a reply. The replicated service is implemented by n replicas. Clients and replicas are non-faulty if they follow the algorithm in Section 4 and if no attacker can forge their signature.

The algorithm provides both *safety* and *liveness* assuming no more than $\lfloor \frac{n-1}{3} \rfloor$ replicas are faulty. Safety means that the replicated service satisfies linearizability [14] (modified to account for Byzantine-faulty clients [4]): it behaves like a centralized implementation that executes operations atomically one at a time. Safety requires the bound on the number of faulty replicas because a faulty replica can behave arbitrarily, e.g., it can destroy its state.

Safety is provided regardless of how many faulty clients are using the service (even if they collude with faulty replicas): all operations performed by faulty clients are observed in a consistent way by non-faulty clients. In particular, if the service operations are designed to preserve some invariants on the service state, faulty clients cannot break those invariants.

The safety property is insufficient to guard against faulty clients, e.g., in a file system a faulty client can write garbage data to some shared file. However, we limit the amount of damage a faulty client can do by providing access control: we authenticate clients and deny access if the client issuing a request does not have the right to invoke the operation. Also, services may provide operations to change the access permissions for a client. Since the algorithm ensures that the effects of access revocation operations are observed consistently by all clients, this provides a powerful mechanism to recover from attacks by faulty clients.

The algorithm does not rely on synchrony to provide safety. Therefore, it must rely on synchrony to provide liveness; otherwise it could be used to implement consensus in an asynchronous system, which is not possible [9]. We guarantee liveness, i.e., clients eventually receive replies to their requests, provided at most $\lfloor \frac{n-1}{3} \rfloor$ replicas are faulty and $delay(t)$ does not

grow faster than t indefinitely. Here, *delay*(t) is the time between the moment t when a message is sent for the first time and the moment when it is received by its destination (assuming the sender keeps retransmitting the message until it is received). (A more precise definition can be found in [4].) This is a rather weak synchrony assumption that is likely to be true in any real system provided network faults are eventually repaired, yet it enables us to circumvent the impossibility result in [9].

The resiliency of our algorithm is optimal: $3f+1$ is the minimum number of replicas that allow an asynchronous system to provide the safety and liveness properties when up to f replicas are faulty (see [2] for a proof). This many replicas are needed because it must be possible to proceed after communicating with $n-f$ replicas, since f replicas might be faulty and not responding. However, it is possible that the f replicas that did not respond are not faulty and, therefore, f of those that responded might be faulty. Even so, there must still be enough responses that those from non-faulty replicas outnumber those from faulty ones, i.e., $n-2f > f$. Therefore $n > 3f$.

The algorithm does not address the problem of fault-tolerant privacy: a faulty replica may leak information to an attacker. It is not feasible to offer fault-tolerant privacy in the general case because service operations may perform arbitrary computations using their arguments and the service state; replicas need this information in the clear to execute such operations efficiently. It is possible to use secret sharing schemes [35] to obtain privacy even in the presence of a threshold of malicious replicas [13] for the arguments and portions of the state that are opaque to the service operations. We plan to investigate these techniques in the future.

4 The Algorithm

Our algorithm is a form of *state machine* replication [17, 34]: the service is modeled as a state machine that is replicated across different nodes in a distributed system. Each state machine replica maintains the service state and implements the service operations. We denote the set of replicas by $\mathcal{R}$ and identify each replica using an integer in $\{0, ..., |\mathcal{R}|-1\}$. For simplicity, we assume $|\mathcal{R}| = 3f+1$ where f is the maximum number of replicas that may be faulty; although there could be more than $3f+1$ replicas, the additional replicas degrade performance (since more and bigger messages are being exchanged) without providing improved resiliency.

The replicas move through a succession of configurations called *views*. In a view one replica is the *primary* and the others are *backups*. Views are numbered consecutively. The primary of a view is replica p such that $p = v \bmod |\mathcal{R}|$, where v is the view number. View changes are carried out when it appears that the primary has failed. Viewstamped Replication [26] and Paxos [18] used a similar approach to tolerate benign faults (as discussed in Section 8.)

The algorithm works roughly as follows:

1. A client sends a request to invoke a service operation to the primary
2. The primary multicasts the request to the backups
3. Replicas execute the request and send a reply to the client
4. The client waits for $f+1$ replies from different replicas with the same result; this is the result of the operation.

Like all state machine replication techniques [34], we impose two requirements on replicas: they must be *deterministic* (i.e., the execution of an operation in a given state and with a given set of arguments must always produce the same result) and they must start in the same state. Given these two requirements, the algorithm ensures the safety property by guaranteeing that *all non-faulty replicas agree on a total order for the execution of requests despite failures.*

The remainder of this section describes a simplified version of the algorithm. We omit discussion of how nodes recover from faults due to lack of space. We also omit details related to message retransmissions. Furthermore, we assume that message authentication is achieved using digital signatures rather than the more efficient scheme based on message authentication codes; Section 5 discusses this issue further. A detailed formalization of the algorithm using the I/O automaton model [21] is presented in [4].

4.1 The Client

A client c requests the execution of state machine operation o by sending a $\langle \text{REQUEST}, o, t, c \rangle_{\sigma_c}$ message to the primary. Timestamp t is used to ensure *exactly-once* semantics for the execution of client requests. Timestamps for c's requests are totally ordered such that later requests have higher timestamps than earlier ones; for example, the timestamp could be the value of the client's local clock when the request is issued.

Each message sent by the replicas to the client includes the current view number, allowing the client to track the view and hence the current primary. A client sends a request to what it believes is the current primary using a point-to-point message. The primary atomically multicasts the request to all the backups using the protocol described in the next section.

A replica sends the reply to the request directly to the client. The reply has the form $\langle \text{REPLY}, v, t, c, i, r \rangle_{\sigma_i}$ where v is the current view number, t is the timestamp of the corresponding request, i is the replica number, and r is the result of executing the requested operation.

The client waits for $f+1$ replies with valid signatures from different replicas, and with the same t and r, before

accepting the result r. This ensures that the result is valid, since at most f replicas can be faulty.

If the client does not receive replies soon enough, it broadcasts the request to all replicas. If the request has already been processed, the replicas simply re-send the reply; replicas remember the last reply message they sent to each client. Otherwise, if the replica is not the primary, it relays the request to the primary. If the primary does not multicast the request to the group, it will eventually be suspected to be faulty by enough replicas to cause a view change.

In this paper we assume that the client waits for one request to complete before sending the next one. But we can allow a client to make asynchronous requests, yet preserve ordering constraints on them.

4.2 Normal-Case Operation

The state of each replica includes the state of the service, a *message log* containing messages the replica has accepted, and an integer denoting the replica's current view. We describe how to truncate the log in Section 4.3.

When the primary, p, receives a client request, m, it starts a three-phase protocol to atomically multicast the request to the replicas. The primary starts the protocol immediately unless the number of messages for which the protocol is in progress exceeds a given maximum. In this case, it buffers the request. Buffered requests are multicast later as a group to cut down on message traffic and CPU overheads under heavy load; this optimization is similar to a group commit in transactional systems [11]. For simplicity, we ignore this optimization in the description below.

The three phases are *pre-prepare*, *prepare*, and *commit*. The pre-prepare and prepare phases are used to totally order requests sent in the same view even when the primary, which proposes the ordering of requests, is faulty. The prepare and commit phases are used to ensure that requests that commit are totally ordered across views.

In the pre-prepare phase, the primary assigns a sequence number, n, to the request, multicasts a pre-prepare message with m piggybacked to all the backups, and appends the message to its log. The message has the form $\langle\langle \text{PRE-PREPARE}, v, n, d\rangle_{\sigma_p}, m\rangle$, where v indicates the view in which the message is being sent, m is the client's request message, and d is m's digest.

Requests are not included in pre-prepare messages to keep them small. This is important because pre-prepare messages are used as a proof that the request was assigned sequence number n in view v in view changes. Additionally, it decouples the protocol to totally order requests from the protocol to transmit the request to the replicas; allowing us to use a transport optimized for small messages for protocol messages and a transport optimized for large messages for large requests.

A backup accepts a pre-prepare message provided:

- the signatures in the request and the pre-prepare message are correct and d is the digest for m;
- it is in view v;
- it has not accepted a pre-prepare message for view v and sequence number n containing a different digest;
- the sequence number in the pre-prepare message is between a low water mark, h, and a high water mark, H.

The last condition prevents a faulty primary from exhausting the space of sequence numbers by selecting a very large one. We discuss how H and h advance in Section 4.3.

If backup i accepts the $\langle\langle \text{PRE-PREPARE}, v, n, d\rangle_{\sigma_p}, m\rangle$ message, it enters the *prepare* phase by multicasting a $\langle \text{PREPARE}, v, n, d, i\rangle_{\sigma_i}$ message to all other replicas and adds both messages to its log. Otherwise, it does nothing.

A replica (including the primary) accepts prepare messages and adds them to its log provided their signatures are correct, their view number equals the replica's current view, and their sequence number is between h and H.

We define the predicate *prepared*(m, v, n, i) to be true if and only if replica i has inserted in its log: the request m, a pre-prepare for m in view v with sequence number n, and $2f$ prepares from different backups that match the pre-prepare. The replicas verify whether the prepares match the pre-prepare by checking that they have the same view, sequence number, and digest.

The pre-prepare and prepare phases of the algorithm guarantee that non-faulty replicas agree on a total order for the requests within a view. More precisely, they ensure the following invariant: if *prepared*(m, v, n, i) is true then *prepared*(m', v, n, j) is false for any non-faulty replica j (including $i = j$) and any m' such that $D(m') \neq D(m)$. This is true because *prepared*(m, v, n, i) and $|\mathcal{R}| = 3f + 1$ imply that at least $f + 1$ non-faulty replicas have sent a pre-prepare or prepare for m in view v with sequence number n. Thus, for *prepared*(m', v, n, j) to be true at least one of these replicas needs to have sent two conflicting prepares (or pre-prepares if it is the primary for v), i.e., two prepares with the same view and sequence number and a different digest. But this is not possible because the replica is not faulty. Finally, our assumption about the strength of message digests ensures that the probability that $m \neq m'$ and $D(m) = D(m')$ is negligible.

Replica i multicasts a $\langle \text{COMMIT}, v, n, D(m), i\rangle_{\sigma_i}$ to the other replicas when *prepared*(m, v, n, i) becomes true. This starts the commit phase. Replicas accept commit messages and insert them in their log provided they are properly signed, the view number in the message is equal to the replica's current view, and the sequence number is between h and H

We define the *committed* and *committed-local* predicates as follows: *committed*(m, v, n) is true if and only if *prepared*(m, v, n, i) is true for all i in some set of $f+1$ non-faulty replicas; and *committed-local*(m, v, n, i) is true if and only if *prepared*(m, v, n, i) is true and i has accepted $2f + 1$ commits (possibly including its own) from different replicas that match the pre-prepare for m; a commit matches a pre-prepare if they have the same view, sequence number, and digest.

The commit phase ensures the following invariant: if *committed-local*(m, v, n, i) is true for some non-faulty i then *committed*(m, v, n) is true. This invariant and the view-change protocol described in Section 4.4 ensure that non-faulty replicas agree on the sequence numbers of requests that commit locally even if they commit in different views at each replica. Furthermore, it ensures that any request that commits locally at a non-faulty replica will commit at $f + 1$ or more non-faulty replicas eventually.

Each replica i executes the operation requested by m after *committed-local*(m, v, n, i) is true and i's state reflects the sequential execution of all requests with lower sequence numbers. This ensures that all non-faulty replicas execute requests in the same order as required to provide the safety property. After executing the requested operation, replicas send a reply to the client. Replicas discard requests whose timestamp is lower than the timestamp in the last reply they sent to the client to guarantee exactly-once semantics.

We do not rely on ordered message delivery, and therefore it is possible for a replica to commit requests out of order. This does not matter since it keeps the pre-prepare, prepare, and commit messages logged until the corresponding request can be executed.

Figure 1 shows the operation of the algorithm in the normal case of no primary faults. Replica 0 is the primary, replica 3 is faulty, and C is the client.

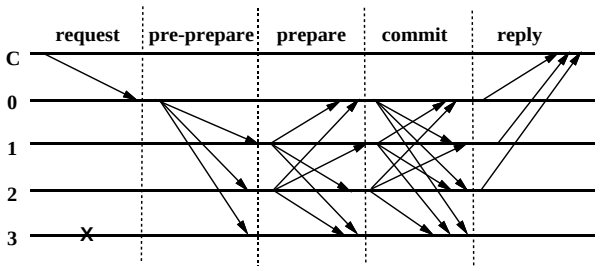


Figure 1: Normal Case Operation

4.3 Garbage Collection

This section discusses the mechanism used to discard messages from the log. For the safety condition to hold, messages must be kept in a replica's log until it knows that the requests they concern have been executed by at least $f + 1$ non-faulty replicas and it can prove this to others in view changes. In addition, if some replica misses messages that were discarded by all non-faulty replicas, it will need to be brought up to date by transferring all or a portion of the service state. Therefore, replicas also need some proof that the state is correct.

Generating these proofs after executing every operation would be expensive. Instead, they are generated periodically, when a request with a sequence number divisible by some constant (e.g., 100) is executed. We will refer to the states produced by the execution of these requests as *checkpoints* and we will say that a checkpoint with a proof is a *stable checkpoint*.

A replica maintains several logical copies of the service state: the last stable checkpoint, zero or more checkpoints that are not stable, and a current state. Copy-on-write techniques can be used to reduce the space overhead to store the extra copies of the state, as discussed in Section 6.3.

The proof of correctness for a checkpoint is generated as follows. When a replica i produces a checkpoint, it multicasts a message $\langle \text{CHECKPOINT}, n, d, i \rangle_{\sigma_i}$ to the other replicas, where n is the sequence number of the last request whose execution is reflected in the state and d is the digest of the state. Each replica collects checkpoint messages in its log until it has $2f + 1$ of them for sequence number n with the same digest d signed by different replicas (including possibly its own such message). These $2f + 1$ messages are the proof of correctness for the checkpoint.

A checkpoint with a proof becomes stable and the replica discards all pre-prepare, prepare, and commit messages with sequence number less than or equal to n from its log; it also discards all earlier checkpoints and checkpoint messages.

Computing the proofs is efficient because the digest can be computed using incremental cryptography [1] as discussed in Section 6.3, and proofs are generated rarely.

The checkpoint protocol is used to advance the low and high water marks (which limit what messages will be accepted). The low-water mark h is equal to the sequence number of the last stable checkpoint. The high water mark $H = h + k$, where k is big enough so that replicas do not stall waiting for a checkpoint to become stable. For example, if checkpoints are taken every 100 requests, k might be 200.

4.4 View Changes

The view-change protocol provides liveness by allowing the system to make progress when the primary fails. View changes are triggered by timeouts that prevent backups from waiting indefinitely for requests to execute. A backup is *waiting* for a request if it received a valid request

and has not executed it. A backup starts a timer when it receives a request and the timer is not already running. It stops the timer when it is no longer waiting to execute the request, but restarts it if at that point it is waiting to execute some other request.

If the timer of backup i expires in view v, the backup starts a view change to move the system to view $v+1$. It stops accepting messages (other than checkpoint, view-change, and new-view messages) and multicasts a $\langle \text{VIEW-CHANGE}, v+1, n, \mathcal{C}, \mathcal{P}, i \rangle_{\sigma_i}$ message to all replicas. Here n is the sequence number of the last stable checkpoint s known to i, $\mathcal{C}$ is a set of $2f+1$ valid checkpoint messages proving the correctness of s, and $\mathcal{P}$ is a set containing a set $\mathcal{P}_m$ for each request m that prepared at i with a sequence number higher than n. Each set $\mathcal{P}_m$ contains a valid pre-prepare message (without the corresponding client message) and $2f$ matching, valid prepare messages signed by different backups with the same view, sequence number, and the digest of m.

When the primary p of view $v+1$ receives $2f$ valid view-change messages for view $v+1$ from other replicas, it multicasts a $\langle \text{NEW-VIEW}, v+1, \mathcal{V}, \mathcal{O} \rangle_{\sigma_p}$ message to all other replicas, where $\mathcal{V}$ is a set containing the valid view-change messages received by the primary plus the view-change message for $v+1$ the primary sent (or would have sent), and $\mathcal{O}$ is a set of pre-prepare messages (without the piggybacked request). $\mathcal{O}$ is computed as follows:

1. The primary determines the sequence number *min-s* of the latest stable checkpoint in $\mathcal{V}$ and the highest sequence number *max-s* in a prepare message in $\mathcal{V}$.
2. The primary creates a new pre-prepare message for view $v+1$ for each sequence number n between *min-s* and *max-s*. There are two cases: (1) there is at least one set in the $\mathcal{P}$ component of some view-change message in $\mathcal{V}$ with sequence number n, or (2) there is no such set. In the first case, the primary creates a new message $\langle \text{PRE-PREPARE}, v+1, n, d \rangle_{\sigma_p}$, where d is the request digest in the pre-prepare message for sequence number n with the highest view number in $\mathcal{V}$. In the second case, it creates a new pre-prepare message $\langle \text{PRE-PREPARE}, v+1, n, d^{null} \rangle_{\sigma_p}$, where d^{null} is the digest of a special *null* request; a null request goes through the protocol like other requests, but its execution is a no-op. (Paxos [18] used a similar technique to fill in gaps.)

Next the primary appends the messages in $\mathcal{O}$ to its log. If *min-s* is greater than the sequence number of its latest stable checkpoint, the primary also inserts the proof of stability for the checkpoint with sequence number *min-s* in its log, and discards information from the log as discussed in Section 4.3. Then it *enters* view $v+1$: at this point it is able to accept messages for view $v+1$.

A backup accepts a new-view message for view $v+1$ if it is signed properly, if the view-change messages it contains are valid for view $v+1$, and if the set $\mathcal{O}$ is correct; it verifies the correctness of $\mathcal{O}$ by performing a computation similar to the one used by the primary to create $\mathcal{O}$. Then it adds the new information to its log as described for the primary, multicasts a prepare for each message in $\mathcal{O}$ to all the other replicas, adds these prepares to its log, and enters view $v+1$.

Thereafter, the protocol proceeds as described in Section 4.2. Replicas redo the protocol for messages between *min-s* and *max-s* but they avoid re-executing client requests (by using their stored information about the last reply sent to each client).

A replica may be missing some request message m or a stable checkpoint (since these are not sent in new-view messages.) It can obtain missing information from another replica. For example, replica i can obtain a missing checkpoint state s from one of the replicas whose checkpoint messages certified its correctness in $\mathcal{V}$. Since $f+1$ of those replicas are correct, replica i will always obtain s or a later certified stable checkpoint. We can avoid sending the entire checkpoint by partitioning the state and stamping each partition with the sequence number of the last request that modified it. To bring a replica up to date, it is only necessary to send it the partitions where it is out of date, rather than the whole checkpoint.

4.5 Correctness

This section sketches the proof that the algorithm provides safety and liveness; details can be found in [4].

4.5.1 Safety

As discussed earlier, the algorithm provides safety if all non-faulty replicas agree on the sequence numbers of requests that commit locally.

In Section 4.2, we showed that if *prepared*(m, v, n, i) is true, *prepared*(m', v, n, j) is false for any non-faulty replica j (including $i = j$) and any m' such that $D(m') \neq D(m)$. This implies that two non-faulty replicas agree on the sequence number of requests that commit locally in the same view at the two replicas.

The view-change protocol ensures that non-faulty replicas also agree on the sequence number of requests that commit locally in different views at different replicas. A request m commits locally at a non-faulty replica with sequence number n in view v only if *committed*(m, v, n) is true. This means that there is a set R_1 containing at least $f+1$ non-faulty replicas such that *prepared*(m, v, n, i) is true for every replica i in the set.

Non-faulty replicas will not accept a pre-prepare for view $v' > v$ without having received a new-view message for v' (since only at that point do they enter the view). But any correct new-view message for view $v' > v$ contains correct view-change messages from every replica i in a

set R_2 of $2f+1$ replicas. Since there are $3f+1$ replicas, R_1 and R_2 must intersect in at least one replica k that is not faulty. k's view-change message will ensure that the fact that m prepared in a previous view is propagated to subsequent views, unless the new-view message contains a view-change message with a stable checkpoint with a sequence number higher than n. In the first case, the algorithm redoes the three phases of the atomic multicast protocol for m with the same sequence number n and the new view number. This is important because it prevents any different request that was assigned the sequence number n in a previous view from ever committing. In the second case no replica in the new view will accept any message with sequence number lower than n. In either case, the replicas will agree on the request that commits locally with sequence number n.

4.5.2 Liveness

To provide liveness, replicas must move to a new view if they are unable to execute a request. But it is important to maximize the period of time when at least $2f+1$ non-faulty replicas are in the same view, and to ensure that this period of time increases exponentially until some requested operation executes. We achieve these goals by three means.

First, to avoid starting a view change too soon, a replica that multicasts a view-change message for view $v+1$ waits for $2f+1$ view-change messages for view $v+1$ and then starts its timer to expire after some time T. If the timer expires before it receives a valid new-view message for $v+1$ or before it executes a request in the new view that it had not executed previously, it starts the view change for view $v+2$ but this time it will wait $2T$ before starting a view change for view $v+3$.

Second, if a replica receives a set of $f+1$ valid view-change messages from other replicas for views greater than its current view, it sends a view-change message for the smallest view in the set, even if its timer has not expired; this prevents it from starting the next view change too late.

Third, faulty replicas are unable to impede progress by forcing frequent view changes. A faulty replica cannot cause a view change by sending a view-change message, because a view change will happen only if at least $f+1$ replicas send view-change messages, but it can cause a view change when it is the primary (by not sending messages or sending bad messages). However, because the primary of view v is the replica p such that $p = v \bmod |\mathcal{R}|$, the primary cannot be faulty for more than f consecutive views.

These three techniques guarantee liveness unless message delays grow faster than the timeout period indefinitely, which is unlikely in a real system.

4.6 Non-Determinism

State machine replicas must be deterministic but many services involve some form of non-determinism. For example, the time-last-modified in NFS is set by reading the server's local clock; if this were done independently at each replica, the states of non-faulty replicas would diverge. Therefore, some mechanism to ensure that all replicas select the same value is needed. In general, the client cannot select the value because it does not have enough information; for example, it does not know how its request will be ordered relative to concurrent requests by other clients. Instead, the primary needs to select the value either independently or based on values provided by the backups.

If the primary selects the non-deterministic value independently, it concatenates the value with the associated request and executes the three phase protocol to ensure that non-faulty replicas agree on a sequence number for the request and value. This prevents a faulty primary from causing replica state to diverge by sending different values to different replicas. However, a faulty primary might send the same, incorrect, value to all replicas. Therefore, replicas must be able to decide deterministically whether the value is correct (and what to do if it is not) based only on the service state.

This protocol is adequate for most services (including NFS) but occasionally replicas must participate in selecting the value to satisfy a service's specification. This can be accomplished by adding an extra phase to the protocol: the primary obtains authenticated values proposed by the backups, concatenates $2f+1$ of them with the associated request, and starts the three phase protocol for the concatenated message. Replicas choose the value by a deterministic computation on the $2f+1$ values and their state, e.g., taking the median. The extra phase can be optimized away in the common case. For example, if replicas need a value that is "close enough" to that of their local clock, the extra phase can be avoided when their clocks are synchronized within some delta.

5 Optimizations

This section describes some optimizations that improve the performance of the algorithm during normal-case operation. All the optimizations preserve the liveness and safety properties.

5.1 Reducing Communication

We use three optimizations to reduce the cost of communication. The first avoids sending most large replies. A client request designates a replica to send the result; all other replicas send replies containing just the digest of the result. The digests allow the client to check the correctness of the result while reducing network

bandwidth consumption and CPU overhead significantly for large replies. If the client does not receive a correct result from the designated replica, it retransmits the request as usual, requesting all replicas to send full replies.

The second optimization reduces the number of message delays for an operation invocation from 5 to 4. Replicas execute a request *tentatively* as soon as the prepared predicate holds for the request, their state reflects the execution of all requests with lower sequence number, and these requests are all known to have committed. After executing the request, the replicas send tentative replies to the client. The client waits for $2f + 1$ matching tentative replies. If it receives this many, the request is guaranteed to commit eventually. Otherwise, the client retransmits the request and waits for $f + 1$ non-tentative replies.

A request that has executed tentatively may abort if there is a view change and it is replaced by a *null* request. In this case the replica reverts its state to the last stable checkpoint in the new-view message or to its last checkpointed state (depending on which one has the higher sequence number).

The third optimization improves the performance of read-only operations that do not modify the service state. A client multicasts a read-only request to all replicas. Replicas execute the request immediately in their tentative state after checking that the request is properly authenticated, that the client has access, and that the request is in fact read-only. They send the reply only after all requests reflected in the tentative state have committed; this is necessary to prevent the client from observing uncommitted state. The client waits for $2f+1$ replies from different replicas with the same result. The client may be unable to collect $2f+1$ such replies if there are concurrent writes to data that affect the result; in this case, it retransmits the request as a regular read-write request after its retransmission timer expires.

5.2 Cryptography

In Section 4, we described an algorithm that uses digital signatures to authenticate all messages. However, we actually use digital signatures only for view-change and new-view messages, which are sent rarely, and authenticate all other messages using message authentication codes (MACs). This eliminates the main performance bottleneck in previous systems [29, 22].

However, MACs have a fundamental limitation relative to digital signatures — the inability to prove that a message is authentic to a third party. The algorithm in Section 4 and previous Byzantine-fault-tolerant algorithms [31, 16] for state machine replication rely on the extra power of digital signatures. We modified our algorithm to circumvent the problem by taking advantage of specific invariants, e.g, the invariant that no two different requests prepare with the same view and sequence number at two non-faulty replicas. The modified algorithm is described in [5]. Here we sketch the main implications of using MACs.

MACs can be computed three orders of magnitude faster than digital signatures. For example, a 200MHz Pentium Pro takes 43ms to generate a 1024-bit modulus RSA signature of an MD5 digest and 0.6ms to verify the signature [37], whereas it takes only 10.3μs to compute the MAC of a 64-byte message on the same hardware in our implementation. There are other public-key cryptosystems that generate signatures faster, e.g., elliptic curve public-key cryptosystems, but signature verification is slower [37] and in our algorithm each signature is verified many times.

Each node (including active clients) shares a 16-byte secret session key with each replica. We compute message authentication codes by applying MD5 to the concatenation of the message with the secret key. Rather than using the 16 bytes of the final MD5 digest, we use only the 10 least significant bytes. This truncation has the obvious advantage of reducing the size of MACs and it also improves their resilience to certain attacks [27]. This is a variant of the secret suffix method [36], which is secure as long as MD5 is collision resistant [27, 8].

The digital signature in a reply message is replaced by a single MAC, which is sufficient because these messages have a single intended recipient. The signatures in all other messages (including client requests but excluding view changes) are replaced by vectors of MACs that we call *authenticators*. An authenticator has an entry for every replica other than the sender; each entry is the MAC computed with the key shared by the sender and the replica corresponding to the entry.

The time to verify an authenticator is constant but the time to generate one grows linearly with the number of replicas. This is not a problem because we do not expect to have a large number of replicas and there is a huge performance gap between MAC and digital signature computation. Furthermore, we compute authenticators efficiently; MD5 is applied to the message once and the resulting context is used to compute each vector entry by applying MD5 to the corresponding session key. For example, in a system with 37 replicas (i.e., a system that can tolerate 12 simultaneous faults) an authenticator can still be computed much more than two orders of magnitude faster than a 1024-bit modulus RSA signature.

The size of authenticators grows linearly with the number of replicas but it grows slowly: it is equal to $30 \times \lfloor \frac{n-1}{3} \rfloor$ bytes. An authenticator is smaller than an RSA signature with a 1024-bit modulus for $n \leq 13$ (i.e., systems that can tolerate up to 4 simultaneous faults), which we expect to be true in most configurations.

6 Implementation

This section describes our implementation. First we discuss the replication library, which can be used as a basis for any replicated service. In Section 6.2 we describe how we implemented a replicated NFS on top of the replication library. Then we describe how we maintain checkpoints and compute checkpoint digests efficiently.

6.1 The Replication Library

The client interface to the replication library consists of a single procedure, *invoke*, with one argument, an input buffer containing a request to invoke a state machine operation. The *invoke* procedure uses our protocol to execute the requested operation at the replicas and select the correct reply from among the replies of the individual replicas. It returns a pointer to a buffer containing the operation result.

On the server side, the replication code makes a number of upcalls to procedures that the server part of the application must implement. There are procedures to execute requests (*execute*), to maintain checkpoints of the service state (*make_checkpoint*, *delete_checkpoint*), to obtain the digest of a specified checkpoint (*get_digest*), and to obtain missing information (*get_checkpoint*, *set_checkpoint*). The *execute* procedure receives as input a buffer containing the requested operation, executes the operation, and places the result in an output buffer. The other procedures are discussed further in Sections 6.3 and 6.4.

Point-to-point communication between nodes is implemented using UDP, and multicast to the group of replicas is implemented using UDP over IP multicast [7]. There is a single IP multicast group for each service, which contains all the replicas. These communication protocols are unreliable; they may duplicate or lose messages or deliver them out of order.

The algorithm tolerates out-of-order delivery and rejects duplicates. View changes can be used to recover from lost messages, but this is expensive and therefore it is important to perform retransmissions. During normal operation recovery from lost messages is driven by the receiver: backups send negative acknowledgments to the primary when they are out of date and the primary retransmits pre-prepare messages after a long timeout. A reply to a negative acknowledgment may include both a portion of a stable checkpoint and missing messages. During view changes, replicas retransmit view-change messages until they receive a matching new-view message or they move on to a later view.

The replication library does not implement view changes or retransmissions at present. This does not compromise the accuracy of the results given in Section 7 because the rest of the algorithm is completely implemented (including the manipulation of the timers that trigger view changes) and because we have formalized the complete algorithm and proved its correctness [4].

6.2 BFS: A Byzantine-Fault-tolerant File System

We implemented BFS, a Byzantine-fault-tolerant NFS service, using the replication library. Figure 2 shows the architecture of BFS. We opted not to modify the kernel NFS client and server because we did not have the sources for the Digital Unix kernel.

A file system exported by the fault-tolerant NFS service is mounted on the client machine like any regular NFS file system. Application processes run unmodified and interact with the mounted file system through the NFS client in the kernel. We rely on user level *relay* processes to mediate communication between the standard NFS client and the replicas. A relay receives NFS protocol requests, calls the *invoke* procedure of our replication library, and sends the result back to the NFS client.

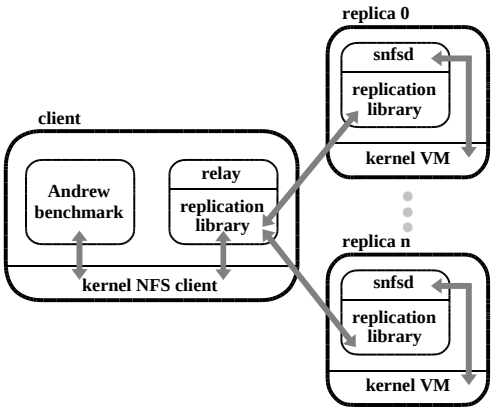


Figure 2: Replicated File System Architecture.

Each replica runs a user-level process with the replication library and our NFS V2 daemon, which we will refer to as *snfsd* (for simple *nfsd*). The replication library receives requests from the relay, interacts with *snfsd* by making upcalls, and packages NFS replies into replication protocol replies that it sends to the relay.

We implemented *snfsd* using a fixed-size memory-mapped file. All the file system data structures, e.g., inodes, blocks and their free lists, are in the mapped file. We rely on the operating system to manage the cache of memory-mapped file pages and to write modified pages to disk asynchronously. The current implementation uses 8KB blocks and inodes contain the NFS status information plus 256 bytes of data, which is used to store directory entries in directories, pointers to blocks in files, and text in symbolic links. Directories and files may also use indirect blocks in a way similar to Unix.

Our implementation ensures that all state machine

replicas start in the same initial state and are deterministic, which are necessary conditions for the correctness of a service implemented using our protocol. The primary proposes the values for time-last-modified and time-last-accessed, and replicas select the larger of the proposed value and one greater than the maximum of all values selected for earlier requests. We do not require synchronous writes to implement NFS V2 protocol semantics because BFS achieves stability of modified data and meta-data through replication [20].

6.3 Maintaining Checkpoints

This section describes how *snfsd* maintains checkpoints of the file system state. Recall that each replica maintains several logical copies of the state: the current state, some number of checkpoints that are not yet stable, and the last stable checkpoint.

snfsd executes file system operations directly in the memory mapped file to preserve locality, and it uses copy-on-write to reduce the space and time overhead associated with maintaining checkpoints. *snfsd* maintains a copy-on-write bit for every 512-byte block in the memory mapped file. When the replication code invokes the *make_checkpoint* upcall, *snfsd* sets all the copy-on-write bits and creates a (volatile) checkpoint record, containing the current sequence number, which it receives as an argument to the upcall, and a list of blocks. This list contains the copies of the blocks that were modified since the checkpoint was taken, and therefore, it is initially empty. The record also contains the digest of the current state; we discuss how the digest is computed in Section 6.4.

When a block of the memory mapped file is modified while executing a client request, *snfsd* checks the copy-on-write bit for the block and, if it is set, stores the block's current contents and its identifier in the checkpoint record for the last checkpoint. Then, it overwrites the block with its new value and resets its copy-on-write bit. *snfsd* retains a checkpoint record until told to discard it via a *delete_checkpoint* upcall, which is made by the replication code when a later checkpoint becomes stable.

If the replication code requires a checkpoint to send to another replica, it calls the *get_checkpoint* upcall. To obtain the value for a block, *snfsd* first searches for the block in the checkpoint record of the stable checkpoint, and then searches the checkpoint records of any later checkpoints. If the block is not in any checkpoint record, it returns the value from the current state.

The use of the copy-on-write technique and the fact that we keep at most 2 checkpoints ensure that the space and time overheads of keeping several logical copies of the state are low. For example, in the Andrew benchmark experiments described in Section 7, the average checkpoint record size is only 182 blocks with a maximum of 500.

6.4 Computing Checkpoint Digests

snfsd computes a digest of a checkpoint state as part of a *make_checkpoint* upcall. Although checkpoints are only taken occasionally, it is important to compute the state digest incrementally because the state may be large. *snfsd* uses an incremental collision-resistant one-way hash function called AdHash [1]. This function divides the state into fixed-size blocks and uses some other hash function (e.g., MD5) to compute the digest of the string obtained by concatenating the block index with the block value for each block. The digest of the state is the sum of the digests of the blocks modulo some large integer. In our current implementation, we use the 512-byte blocks from the copy-on-write technique and compute their digest using MD5.

To compute the digest for the state incrementally, *snfsd* maintains a table with a hash value for each 512-byte block. This hash value is obtained by applying MD5 to the block index concatenated with the block value at the time of the last checkpoint. When *make_checkpoint* is called, *snfsd* obtains the digest d for the previous checkpoint state (from the associated checkpoint record). It computes new hash values for each block whose copy-on-write bit is reset by applying MD5 to the block index concatenated with the current block value. Then, it adds the new hash value to d, subtracts the old hash value from d, and updates the table to contain the new hash value. This process is efficient provided the number of modified blocks is small; as mentioned above, on average 182 blocks are modified per checkpoint for the Andrew benchmark.

7 Performance Evaluation

This section evaluates the performance of our system using two benchmarks: a micro-benchmark and the Andrew benchmark [15]. The micro-benchmark provides a service-independent evaluation of the performance of the replication library; it measures the latency to invoke a null operation, i.e., an operation that does nothing.

The Andrew benchmark is used to compare BFS with two other file systems: one is the NFS V2 implementation in Digital Unix, and the other is identical to BFS except without replication. The first comparison demonstrates that our system is practical by showing that its latency is similar to the latency of a commercial system that is used daily by many users. The second comparison allows us to evaluate the overhead of our algorithm accurately within an implementation of a real service.

7.1 Experimental Setup

The experiments measure normal-case behavior (i.e., there are no view changes), because this is the behavior

that determines the performance of the system. All experiments ran with one client running two relay processes, and four replicas. Four replicas can tolerate one Byzantine fault; we expect this reliability level to suffice for most applications. The replicas and the client ran on identical DEC 3000/400 Alpha workstations. These workstations have a 133 MHz Alpha 21064 processor, 128 MB of memory, and run Digital Unix version 4.0. The file system was stored by each replica on a DEC RZ26 disk. All the workstations were connected by a 10Mbit/s switched Ethernet and had DEC LANCE Ethernet interfaces. The switch was a DEC EtherWORKS 8T/TX. The experiments were run on an isolated network.

The interval between checkpoints was 128 requests, which causes garbage collection to occur several times in any of the experiments. The maximum sequence number accepted by replicas in pre-prepare messages was 256 plus the sequence number of the last stable checkpoint.

7.2 Micro-Benchmark

The micro-benchmark measures the latency to invoke a null operation. It evaluates the performance of two implementations of a simple service with no state that implements null operations with arguments and results of different sizes. The first implementation is replicated using our library and the second is unreplicated and uses UDP directly. Table 1 reports the response times measured at the client for both read-only and read-write operations. They were obtained by timing 10,000 operation invocations in three separate runs and we report the median value of the three runs. The maximum deviation from the median was always below 0.3% of the reported value. We denote each operation by *a/b*, where *a* and *b* are the sizes of the operation argument and result in KBytes.

arg./res. (KB)	replicated		without replication
	read-write	read-only	
0/0	3.35 (309%)	1.62 (98%)	0.82
4/0	14.19 (207%)	6.98 (51%)	4.62
0/4	8.01 (72%)	5.94 (27%)	4.66

Table 1: Micro-benchmark results (in milliseconds); the percentage overhead is relative to the unreplicated case.

The overhead introduced by the replication library is due to extra computation and communication. For example, the computation overhead for the read-write 0/0 operation is approximately 1.06ms, which includes 0.55ms spent executing cryptographic operations. The remaining 1.47ms of overhead are due to extra communication; the replication library introduces an extra message round-trip, it sends larger messages, and it increases the number of messages received by each node relative to the service without replication.

The overhead for read-only operations is significantly lower because the optimization discussed in Section 5.1 reduces both computation and communication overheads. For example, the computation overhead for the read-only 0/0 operation is approximately 0.43ms, which includes 0.23ms spent executing cryptographic operations, and the communication overhead is only 0.37ms because the protocol to execute read-only operations uses a single round-trip.

Table 1 shows that the relative overhead is lower for the 4/0 and 0/4 operations. This is because a significant fraction of the overhead introduced by the replication library is independent of the size of operation arguments and results. For example, in the read-write 0/4 operation, the large message (the reply) goes over the network only once (as discussed in Section 5.1) and only the cryptographic overhead to process the reply message is increased. The overhead is higher for the read-write 4/0 operation because the large message (the request) goes over the network twice and increases the cryptographic overhead for processing both request and pre-prepare messages.

It is important to note that this micro-benchmark represents the worst case overhead for our algorithm because the operations perform no work and the unreplicated server provides very weak guarantees. Most services will require stronger guarantees, e.g., authenticated connections, and the overhead introduced by our algorithm relative to a server that implements these guarantees will be lower. For example, the overhead of the replication library relative to a version of the unreplicated service that uses MACs for authentication is only 243% for the read-write 0/0 operation and 4% for the read-only 4/0 operation.

We can estimate a rough lower bound on the performance gain afforded by our algorithm relative to Rampart [30]. Reiter reports that Rampart has a latency of 45ms for a multi-RPC of a null message in a 10 Mbit/s Ethernet network of 4 SparcStation 10s [30]. The multi-RPC is sufficient for the primary to invoke a state machine operation but for an arbitrary client to invoke an operation it would be necessary to add an extra message delay and an extra RSA signature and verification to authenticate the client; this would lead to a latency of at least 65ms (using the RSA timings reported in [29].) Even if we divide this latency by 1.7, the ratio of the SPECint92 ratings of the DEC 3000/400 and the SparcStation 10, our algorithm still reduces the latency to invoke the read-write and read-only 0/0 operations by factors of more than 10 and 20, respectively. Note that this scaling is conservative because the network accounts for a significant fraction of Rampart's latency [29] and Rampart's results were obtained using 300-bit modulus RSA signatures, which are not considered secure today unless the keys used to

generate them are refreshed very frequently.

There are no published performance numbers for SecureRing [16] but it would be slower than Rampart because its algorithm has more message delays and signature operations in the critical path.

7.3 Andrew Benchmark

The Andrew benchmark [15] emulates a software development workload. It has five phases: (1) creates subdirectories recursively; (2) copies a source tree; (3) examines the status of all the files in the tree without examining their data; (4) examines every byte of data in all the files; and (5) compiles and links the files.

We use the Andrew benchmark to compare BFS with two other file system configurations: NFS-std, which is the NFS V2 implementation in Digital Unix, and BFS-nr, which is identical to BFS but with no replication. BFS-nr ran two simple UDP relays on the client, and on the server it ran a thin veneer linked with a version of *snfsd* from which all the checkpoint management code was removed. This configuration does *not* write modified file system state to disk before replying to the client. Therefore, it does not implement NFS V2 protocol semantics, whereas both BFS and NFS-std do.

Out of the 18 operations in the NFS V2 protocol only `getattr` is read-only because the time-last-accessed attribute of files and directories is set by operations that would otherwise be read-only, e.g., `read` and `lookup`. The result is that our optimization for read-only operations can rarely be used. To show the impact of this optimization, we also ran the Andrew benchmark on a second version of BFS that modifies the `lookup` operation to be read-only. This modification violates strict Unix file system semantics but is unlikely to have adverse effects in practice.

For all configurations, the actual benchmark code ran at the client workstation using the standard NFS client implementation in the Digital Unix kernel with the same `mount` options. The most relevant of these options for the benchmark are: UDP transport, 4096-byte read and write buffers, allowing asynchronous client writes, and allowing attribute caching.

We report the mean of 10 runs of the benchmark for each configuration. The sample standard deviation for the total time to run the benchmark was always below 2.6% of the reported value but it was as high as 14% for the individual times of the first four phases. This high variance was also present in the NFS-std configuration. The estimated error for the reported mean was below 4.5% for the individual phases and 0.8% for the total.

Table 2 shows the results for BFS and BFS-nr. The comparison between BFS-strict and BFS-nr shows that the overhead of Byzantine fault tolerance for this service is low — BFS-strict takes only 26% more time to run the complete benchmark. The overhead is lower than what was observed for the micro-benchmarks because the client spends a significant fraction of the elapsed time computing between operations, i.e., between receiving the reply to an operation and issuing the next request, and operations at the server perform some computation. But the overhead is not uniform across the benchmark phases. The main reason for this is a variation in the amount of time the client spends computing between operations; the first two phases have a higher relative overhead because the client spends approximately 40% of the total time computing between operations, whereas it spends approximately 70% during the last three phases.

	BFS		
phase	strict	r/o lookup	BFS-nr
1	0.55 (57%)	0.47 (34%)	0.35
2	9.24 (82%)	7.91 (56%)	5.08
3	7.24 (18%)	6.45 (6%)	6.11
4	8.77 (18%)	7.87 (6%)	7.41
5	38.68 (20%)	38.38 (19%)	32.12
total	64.48 (26%)	61.07 (20%)	51.07

Table 2: Andrew benchmark: BFS vs BFS-nr. The times are in seconds.

The table shows that applying the read-only optimization to `lookup` improves the performance of BFS significantly and reduces the overhead relative to BFS-nr to 20%. This optimization has a significant impact in the first four phases because the time spent waiting for `lookup` operations to complete in BFS-strict is at least 20% of the elapsed time for these phases, whereas it is less than 5% of the elapsed time for the last phase.

	BFS		
phase	strict	r/o lookup	NFS-std
1	0.55 (-69%)	0.47 (-73%)	1.75
2	9.24 (-2%)	7.91 (-16%)	9.46
3	7.24 (35%)	6.45 (20%)	5.36
4	8.77 (32%)	7.87 (19%)	6.60
5	38.68 (-2%)	38.38 (-2%)	39.35
total	64.48 (3%)	61.07 (-2%)	62.52

Table 3: Andrew benchmark: BFS vs NFS-std. The times are in seconds.

Table 3 shows the results for BFS vs NFS-std. These results show that BFS can be used in practice — BFS-strict takes only 3% more time to run the complete benchmark. Thus, one could replace the NFS V2 implementation in Digital Unix, which is used daily by many users, by BFS without affecting the latency perceived by those users. Furthermore, BFS with the read-only optimization for the `lookup` operation is actually 2% faster than NFS-std.

The overhead of BFS relative to NFS-std is not the

same for all phases. Both versions of BFS are faster than NFS-std for phases 1, 2, and 5 but slower for the other phases. This is because during phases 1, 2, and 5 a large fraction (between 21% and 40%) of the operations issued by the client are *synchronous*, i.e., operations that require the NFS implementation to ensure stability of modified file system state before replying to the client. NFS-std achieves stability by writing modified state to disk whereas BFS achieves stability with lower latency using replication (as in Harp [20]). NFS-std is faster than BFS (and BFS-nr) in phases 3 and 4 because the client issues no synchronous operations during these phases.

8 Related Work

Most previous work on replication techniques ignored Byzantine faults or assumed a synchronous system model (e.g., [17, 26, 18, 34, 6, 10]). Viewstamped replication [26] and Paxos [18] use views with a primary and backups to tolerate benign faults in an asynchronous system. Tolerating Byzantine faults requires a much more complex protocol with cryptographic authentication, an extra pre-prepare phase, and a different technique to trigger view changes and select primaries. Furthermore, our system uses view changes only to select a new primary but never to select a different set of replicas to form the new view as in [26, 18].

Some agreement and consensus algorithms tolerate Byzantine faults in asynchronous systems (e.g,[2, 3, 24]). However, they do not provide a complete solution for state machine replication, and furthermore, most of them were designed to demonstrate theoretical feasibility and are too slow to be used in practice. Our algorithm during normal-case operation is similar to the Byzantine agreement algorithm in [2] but that algorithm is unable to survive primary failures.

The two systems that are most closely related to our work are Rampart [29, 30, 31, 22] and SecureRing [16]. They implement state machine replication but are more than an order of magnitude slower than our system and, most importantly, they rely on synchrony assumptions.

Both Rampart and SecureRing must exclude faulty replicas from the group to make progress (e.g., to remove a faulty primary and elect a new one), and to perform garbage collection. They rely on failure detectors to determine which replicas are faulty. However, failure detectors cannot be accurate in an asynchronous system [21], i.e., they may misclassify a replica as faulty. Since correctness requires that fewer than $1/3$ of group members be faulty, a misclassification can compromise correctness by removing a non-faulty replica from the group. This opens an avenue of attack: an attacker gains control over a single replica but does not change its behavior in any detectable way; then it slows correct replicas or the communication between them until enough are excluded from the group.

To reduce the probability of misclassification, failure detectors can be calibrated to delay classifying a replica as faulty. However, for the probability to be negligible the delay must be very large, which is undesirable. For example, if the primary has actually failed, the group will be unable to process client requests until the delay has expired. Our algorithm is not vulnerable to this problem because it never needs to exclude replicas from the group.

Phalanx [23, 25] applies quorum replication techniques [12] to achieve Byzantine fault-tolerance in asynchronous systems. This work does not provide generic state machine replication; instead, it offers a data repository with operations to read and write individual variables and to acquire locks. The semantics it provides for read and write operations are weaker than those offered by our algorithm; we can implement arbitrary operations that access any number of variables, whereas in Phalanx it would be necessary to acquire and release locks to execute such operations. There are no published performance numbers for Phalanx but we believe our algorithm is faster because it has fewer message delays in the critical path and because of our use of MACs rather than public key cryptography. The approach in Phalanx offers the potential for improved scalability; each operation is processed by only a subset of replicas. But this approach to scalability is expensive: it requires $n > 4f + 1$ to tolerate f faults; each replica needs a copy of the state; and the load on each replica decreases slowly with n (it is $O(1/\sqrt{n})$).

9 Conclusions

This paper has described a new state-machine replication algorithm that is able to tolerate Byzantine faults and can be used in practice: it is the first to work correctly in an asynchronous system like the Internet and it improves the performance of previous algorithms by more than an order of magnitude.

The paper also described BFS, a Byzantine-fault-tolerant implementation of NFS. BFS demonstrates that it is possible to use our algorithm to implement real services with performance close to that of an unreplicated service — the performance of BFS is only 3% worse than that of the standard NFS implementation in Digital Unix. This good performance is due to a number of important optimizations, including replacing public-key signatures by vectors of message authentication codes, reducing the size and number of messages, and the incremental checkpoint-management techniques.

One reason why Byzantine-fault-tolerant algorithms will be important in the future is that they can allow systems to continue to work correctly even when there are software errors. Not all errors are survivable; our approach cannot mask a software error that occurs

at all replicas. However, it can mask errors that occur independently at different replicas, including nondeterministic software errors, which are the most problematic and persistent errors since they are the hardest to detect. In fact, we encountered such a software bug while running our system, and our algorithm was able to continue running correctly in spite of it.

There is still much work to do on improving our system. One problem of special interest is reducing the amount of resources required to implement our algorithm. The number of replicas can be reduced by using f replicas as witnesses that are involved in the protocol only when some full replica fails. We also believe that it is possible to reduce the number of copies of the state to $f + 1$ but the details remain to be worked out.

Acknowledgments

We would like to thank Atul Adya, Chandrasekhar Boyapati, Nancy Lynch, Sape Mullender, Andrew Myers, Liuba Shrira, and the anonymous referees for their helpful comments on drafts of this paper.

References

[1] M. Bellare and D. Micciancio. A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost. In *Advances in Cryptology – Eurocrypt 97*, 1997.

[2] G. Bracha and S. Toueg. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM*, 32(4), 1995.

[3] R. Canneti and T. Rabin. Optimal Asynchronous Byzantine Agreement. Technical Report #92-15, Computer Science Department, Hebrew University, 1992.

[4] M. Castro and B. Liskov. A Correctness Proof for a Practical Byzantine-Fault-Tolerant Replication Algorithm. Technical Memo MIT/LCS/TM-590, MIT Laboratory for Computer Science, 1999.

[5] M. Castro and B. Liskov. Authenticated Byzantine Fault Tolerance Without Public-Key Cryptography. Technical Memo MIT/LCS/TM-589, MIT Laboratory for Computer Science, 1999.

[6] F. Cristian, H. Aghili, H. Strong, and D. Dolev. Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. In *International Conference on Fault Tolerant Computing*, 1985.

[7] S. Deering and D. Cheriton. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems*, 8(2), 1990.

[8] H. Dobbertin. The Status of MD5 After a Recent Attack. *RSA Laboratories' CryptoBytes*, 2(2), 1996.

[9] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus With One Faulty Process. *Journal of the ACM*, 32(2), 1985.

[10] J. Garay and Y. Moses. Fully Polynomial Byzantine Agreement for n >3t Processors in t+1 Rounds. *SIAM Journal of Computing*, 27(1), 1998.

[11] D. Gawlick and D. Kinkade. Varieties of Concurrency Control in IMS/VS Fast Path. *Database Engineering*, 8(2), 1985.

[12] D. Gifford. Weighted Voting for Replicated Data. In *Symposium on Operating Systems Principles*, 1979.

[13] M. Herlihy and J. Tygar. How to make replicated data secure. *Advances in Cryptology (LNCS 293)*, 1988.

[14] M. Herlihy and J. Wing. Axioms for Concurrent Objects. In *ACM Symposium on Principles of Programming Languages*, 1987.

[15] J. Howard et al. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1), 1988.

[16] K. Kihlstrom, L. Moser, and P. Melliar-Smith. The SecureRing Protocols for Securing Group Communication. In *Hawaii International Conference on System Sciences*, 1998.

[17] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7), 1978.

[18] L. Lamport. The Part-Time Parliament. Technical Report 49, DEC Systems Research Center, 1989.

[19] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.

[20] B. Liskov et al. Replication in the Harp File System. In *ACM Symposium on Operating System Principles*, 1991.

[21] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, 1996.

[22] D. Malkhi and M. Reiter. A High-Throughput Secure Reliable Multicast Protocol. In *Computer Security Foundations Workshop*, 1996.

[23] D. Malkhi and M. Reiter. Byzantine Quorum Systems. In *ACM Symposium on Theory of Computing*, 1997.

[24] D. Malkhi and M. Reiter. Unreliable Intrusion Detection in Distributed Computations. In *Computer Security Foundations Workshop*, 1997.

[25] D. Malkhi and M. Reiter. Secure and Scalable Replication in Phalanx. In *IEEE Symposium on Reliable Distributed Systems*, 1998.

[26] B. Oki and B. Liskov. Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems. In *ACM Symposium on Principles of Distributed Computing*, 1988.

[27] B. Preneel and P. Oorschot. MDx-MAC and Building Fast MACs from Hash Functions. In *Crypto 95*, 1995.

[28] C. Pu, A. Black, C. Cowan, and J. Walpole. A Specialization Toolkit to Increase the Diversity of Operating Systems. In *ICMAS Workshop on Immunity-Based Systems*, 1996.

[29] M. Reiter. Secure Agreement Protocols. In *ACM Conference on Computer and Communication Security*, 1994.

[30] M. Reiter. The Rampart Toolkit for Building High-Integrity Services. *Theory and Practice in Distributed Systems (LNCS 938)*, 1995.

[31] M. Reiter. A Secure Group Membership Protocol. *IEEE Transactions on Software Engineering*, 22(1), 1996.

[32] R. Rivest. The MD5 Message-Digest Algorithm. Internet RFC-1321, 1992.

[33] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2), 1978.

[34] F. Schneider. Implementing Fault-Tolerant Services Using The State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4), 1990.

[35] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11), 1979.

[36] G. Tsudik. Message Authentication with One-Way Hash Functions. *ACM Computer Communications Review*, 22(5), 1992.

[37] M. Wiener. Performance Comparison of Public-Key Cryptosystems. *RSA Laboratories' CryptoBytes*, 4(1), 1998.

Proof of Storage-Time: Efficiently Checking Continuous Data Availability*

Giuseppe Ateniese[1], Long Chen[2,3], Mohammad Etemad[1], and Qiang Tang[2,3]

[1] Stevens Institute of Technology
{gatenies, metemad}@stevens.edu
[2] New Jersey Institute of Technology
{longchen, qiang}@njit.edu
[3] JDD-NJIT-ISCAS Joint Blockchain Lab

Abstract. A high-quality outsourced storage service is crucial for many existing applications. For example, hospitals and data centers need to guarantee the availability of their systems to perform routine daily activities. Such a system should protect users against downtime and ensure data availability over time. Continuous data availability is a critical property to measure the quality of an outsourced storage service, which implies that outsourced data is continuously available to the server during the entire storage period. We formally study the Proof of Storage-Time (PoSt), the notion initially proposed in the Filecoin whitepaper, which enables a verifier to audit the continuous data availability of an outsourced storage service. We provide a formal security model of PoSt and generic constructions that are proven secure under our definition. Moreover, our concrete instantiation can yield a PoSt protocol with an extremely efficient verification: a single hash computation to verify a proof of size around 200 bits. This makes our scheme applicable even in the decentralized storage marketplace enabled by blockchain.

Keywords: Outsourced storage · Continuous availability · Proof of storage.

1 Introduction

Outsourced storage has become a common practice over the years for backup, data sharing, and more. Increasingly enterprises and individuals choose to rely on a cloud service provider, for example, Amazon Simple Storage Service (S3), to store and maintain their data. If the server collapses, as several occurrences reported recently, e.g., [46, 2], there would be severe repercussions hindering business operations, causing productivity decrease, customer dissatisfaction, or even revenue reduction. The consequences are particularly dire for certain applications that need to be run 24/7 [10]. Therefore, a system that ensures consistent service availability is highly desirable for those mission- and business-critical applications.

Continuous availability, on the other hand, becomes one distinguishing feature that several major storage providers, such as DELL EMC [43] and IBM [39], readily advertise. How to provide such a property from a system perspective has been intensively studied by various researchers [19, 29, 34]. In general, continuous availability should protect users against downtime, whatever the cause, and ensures that files remain accessible to users anytime and anywhere.

Continuous data availability or possession is an enhanced storage integrity feature that is difficult to achieve. To provide a highly reliable service, cloud providers have to deal with all kinds of failures, power outages, or even attacks on their servers. They have to include more replications, devote more redundant hardware and software components, and handle complex administrative workloads. It is thus straightforward to recognize that providing continuous data availability is costly and forces cloud storage providers to adopt specialized hardware and software solutions [12]. A dishonest provider would likely offload those burdens and provide an inferior service without continuous data availability. Moreover, data owners are charged more for reliable storage [35] and may require an irrefutable guarantee that the service they are paying for is being provided correctly. An immediate question arises: how can we verify that a storage provider is indeed supplying continuous data availability, i.e., the provider is virtually always in possession of the outsourced data so that the data owner can retrieve it at any time?

* A preliminary version of this paper appeared at NDSS 2020. The authors are listed in alphabetical order.

Similar to other data availability techniques, such as proof of data possession (PDP) [8] and proof of retrievability (PoR) [31], the verification procedure should be as efficient as possible, both in terms of computation and communication. Ideally, those costs should be independent of the data size and the length of time for continuous storage.

We strive for a very light verification procedure also due to the emerging application of decentralized storage marketplaces powered by Ethereum [49] or Filecoin [40]. This new storage framework is believed to provide superior resilience and reduce costs [16]. In such a setting, data owners simply publish a smart contract that pays storage peer nodes once a succinct and valid proof of storage is received. As smart contracts use on-chain messages and can be expensive to run, it's crucial to minimize both the size and computational cost of verification.

The above discussion highlights a central question we would like to study in this article:

How can we efficiently monitor storage providers to ensure outsourced data is continuously available?

Naive attempts. Proof of data possession (PDP) [8] and proof of retrievability (PoR) [31] are cryptographic protocols that enable a client to efficiently verify whether data stored at the cloud provider is intact and available. The original PoR scheme, based on *hidden sentinels*, worked only for encrypted files and a limited number of challenges. PDP, based on *homomorphic tags*, had no such restrictions and offered public verifiability where everybody, not just the data owner, can verify the proofs. The field evolved rapidly and schemes with better efficiency [44, 9, 17, 45, 5], or advanced properties [50, 51, 20] were introduced. New PoR schemes are based on homomorphic tags and can be seen as PDP schemes coupled together with erasure coding. This extra step, while costly, guarantees retrievability of the file [7]. However, PDP/PoR protocols certify data integrity and availability only at the time a valid proof is processed. Between two proofs, there is nothing that can be guaranteed about the availability of the outsourced data. In principle, a rogue storage node could delegate storage contracts to other nodes that may offer inferior service and then recover the data in time to respond to the next challenge. It's possible to request the client to challenge the storage provider frequently, but this method is inefficient in terms of communication and computational cost for the client that must verify multiple proofs. Besides, it requires the client to be always online.

1.1 Our contributions

To address the problem of efficiently verifying continuous data availability, we give a formal and systematic treatment for the new notion of proof of storage-time (PoSt) (Filecoin initially proposed a similar notion called proof of spacetime [40], but the name "spacetime" now refers to an existing primitive [33] so we renamed it to avoid confusion). PoSt is a challenge-response protocol that allows the prover to convince the verifier that data is continuously available and retrievable for a range of time. Efficiency in our context means that the proof from the prover must be succinct, and the verifier does not have to be always online.

In particular, we first formally define the security properties of PoSt, i.e., what *continuous availability* precisely means. We then give a warm-up construction paired with a rigorous security analysis, followed up by our main construction with further efficiency improvements as well as supporting advanced properties such as "public verifiability/validation". We also demonstrate the efficiency of our protocol by implementing it for various choices of the parameters.

Formally defining PoSt. The syntax of PoSt is similar to PoR/ PDP. The verifier sends a challenge and receives a succinct response after a time T specified by the verifier. The verifier can be offline most of the time.

Providing a precise security model is intricate. We need to define the continuous availability requirement formally, i.e., the data is possessed by the prover *at any time* during the storage period. Intuitively, we shall upgrade the soundness definition of PDP/PoR [8, 44], which defines an extractor algorithm (similar to the classical notion of proof of knowledge) that the data can be extracted via (non-black-box) interaction with the prover. To capture continuous availability, we could define a stronger extractor algorithm that can extract the data from the prover at any point in time. But, a critical question arises: how do we ensure the extracted knowledge of data is indeed *presently* possessed by the prover at a specific time?

Informally, the non-black-box PoSt prover is modeled as an interactive Turing machine (ITM); thus, any knowledge/data that is presently possessed by the machine must be either preserved in the configuration (memory) at that time point or hardcoded in the transition function. This allows us to capture "continuous extractability" by requiring the extractor to operate after is provided with *the configuration of the prover's ITM at any specified execution step along with the transition function.*

To make our definition more general, we choose a parameter t to characterize the approximation of the above idealized "continuous extractability", i.e., the extractor is provided by a bunch of configurations that correspond to an epoch with length t instead of one single step. Of course, the smaller t is the better approximation of the continuous availability the model will be. Intuitively, this approximation mimics the trivial solution that the data is audited every epoch with length t.[4]

Non-triviality of the construction. Constructing a PoSt requires special care. Simple improvements over naive attempts may still suffer from various nuisances. For example, the verifier may execute a PoR in each time slot t during the range period T. To relieve the verifier from being always online, he could send all challenges in advance. However, the prover could cheat by computing all PoR proofs rapidly and, thus, spending less time than t for each challenge. On the other hand, if all challenges are sent at the end of the period, the prover could keep data offline for most of the time and then retrieve it when required. So the main challenge is to find a protocol where the prover is challenged often (e.g., once every time slot t) without requiring the verifier to stay online and interact with the prover.

Filecoin proposed a candidate PoSt construction in their whitepaper [40]. Their idea is to let the prover generate a sequence of PDP/PoR proofs, where the challenge for each proof is derived from hashing the antecedent proof in the sequence. In this way, the verifier needs to provide only the first challenge and then can stay put offline.

While the idea is reasonable and intuitive, it does not provide the security guarantee needed for a PoSt protocol. The main issue is that the prover can run proof-of-storage protocols much faster than expected or estimated by the verifier. Once all proofs are computed, a malicious storage provider could simply put data offline until it is rechallenged. This is a severe drawback since proof-of-storage schemes can be accelerated through parallelism.

Warm-up construction. The time constraints of PoSt are quite strict and critical. One way to build a PoSt protocol that can be proven secure is to leverage the recently proposed notion of verifiable delay function (VDF). In a verifiable delay function, the evaluation of the function must be delayed by a specified amount of time (currently measured by the number of certain unit operations), but the results can be verified much more efficiently. More importantly, the delay holds even if one uses a parallel computer. With the help of a VDF, we could now *compel* the storage provider to generate a PDP/PoR proof in every time slot with length t.

Intuitively, we require the prover to generate the challenge for each PDP/PoR instance from the output of a VDF. Concretely, each challenge $c_i = \mathcal{H}\left(\mathsf{VDF.Eval}(\mathcal{G}(p_{i-1}))\right)$, where p_{i-1} is the antecedent PDP/PoR proof, $\mathcal{G}(\cdot), \mathcal{H}(\cdot)$ are properly-chosen hash functions, and $\mathsf{VDF.Eval}(\cdot)$ is the VDF evaluation algorithm. The prover returns all challenge-proof pairs together with the respective VDF proofs. Recall that the execution time is divided into a specific number of slots. Then, the verifier selects a proper VDF delay time to ensure the prover calculates at least one PDP/PoR proof per time slot. If the prover is not fast enough, the PoSt will not be computed in time.

Striving for efficient verification. The intuitive protocol above is quite simple and provably secure, but it's very inefficient. The communication cost is high, and the verification procedure is computationally expensive. The homomorphic aggregation techniques originally introduced in PDP [8] are not applicable here since the sequentiality of challenges is critical to PoSt.

One of our main innovations is to come up with a different strategy. Our idea is to let the verifier *reproduce* the same sequence of PDP/PoR instances as the prover. Thus, rather than verifying all proofs and VDFs, the verifier must simply check that two sequences of PDP/PoR proofs are the same. The comparison can be efficiently realized through collision-resistant hash functions.

[4] In practice, when t is reasonably small, the cost for the storage provider to frequently move the data back and forth could be even higher than simply keeping them online. On the other hand, there should be a natural trade-off between efficiency and precision, which is expressed by the extra parameter t.

However, we still must overcome two remaining challenges to make our idea practical: (1) the prover algorithm needs to be deterministic to ensure that the reproduction performed by the verifier is the same; (2) the verifier algorithm must be efficient. The first point is simple to address since we can rely on multiple and suitable PDP/PoR candidates [31, 8, 44, 45]. Controlling the computation cost of the verifier is challenging, and we make two additional observations:

- We could leverage an asymmetric VDF in the sense that there is a trapdoor that allows anyone to compute the evaluation function efficiently without any delay. These computations can be moved to the setup phase. Our construction of PoSt can be viewed as a practical application of the notion of trapdoor delay function (TDF) mentioned by Wesolowski [48].
- We could adopt a bounded-use PDP/PoR that supports only a limited number of challenges. Given the structure of our protocol, this is not a limitation and allows us to reduce costs significantly. Indeed, bounded-use proof of storage protocols can be obtained purely from symmetric-key primitives, and enjoy greater efficiency than the algebraic constructions.

We demonstrate our method in Fig. 1.

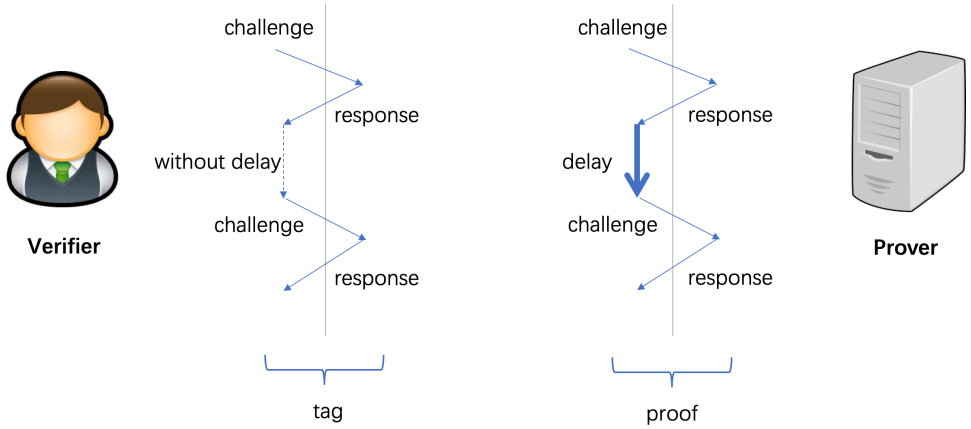


Fig. 1. Structure of our construction

Public validation. In the application of a decentralized storage marketplace as proposed by Filecoin [40], the data owner could crowdsource the storage service and leverage a smart contract that pays providers if they can produce a valid PoSt proof of continuous data availability. Therefore, we could consider the notion of *public validation* where a smart contract or any third party could validate PoSt proofs from public knowledge. An even stronger property, *public verifiability*, insists that the verifier possesses no trapdoor at all, see section 4 for details. Note that the warm-up construction introduced above provides public verifiability while its compact version, which is presented later, provides public validation.

Indeed, consider the following observation. When comparing two sequences of proofs, it's possible to check the digests (hashes) of the two sequences directly. Thus, the verifier could further hash the digest of the sequence to derive d and then make d public or embed it into the smart contract. Anyone with d can check whether $h(\pi) = d$, where $h(\cdot)$ is a collision-resistant hash and π is the PoSt proof. In the end, we obtain a PoSt protocol whose verification algorithm performs only a single hash computation!

An efficient instantiation. Note that in a PoSt, the prover's computational cost is intrinsic since the nature of PoSt requires the prover to access data blocks frequently. Thus, the main concern relevant to efficiency is the setup step. However, since we can use a stateless PoR from highly efficient symmetric-key primitives, the total cost of the setup procedure can also be made practical. According to our experiment results (section 7), the setup procedure for a file of size 64 MB, which is supposed to be stored for 1 month and verified every hour, will take less than 4 minutes. As a comparison, a 64MB-file setup procedure of the outsourced PoR scheme Fortress [5] would cost more than 10 minutes for both the prover and the verifier. Also, note that we did not

perform any optimization, and the main costly component is due to hashing large files (which can be pre-computed and optimized in various ways).

Although VDFs somehow ensure that adversaries cannot evaluate the function much faster, precautions are needed when instantiating the concrete parameters for the VDF evaluations. (1) A (small) gap exists between the number of unit operations needed for an honest evaluator and that for the adversary. We resolve this by selecting parameters to guarantee that the adversary cannot finish the proof ahead of time while the honest evaluator can finish the work within a reasonable extension; (2) The calculation of unit operations could be expedited. Several organizations, including Filecoin and Ethereum, have made substantial investments in finding the fastest machine possible for these tasks. We provide more detailed discussions on these issues in Sec. 6.2.

Security analysis. Rigorously analyzing the security of PoSt turned out to be challenging. For soundness, we need to extract data via interacting with a "partial" legitimate prover, i.e., the only items the extractor gets are a subset of configurations and the code of the transition function. The main difficulty is that it seems impossible to unify the strategies of a cheating prover by only knowing that the final proof is admissible. Thus, it is complicated to recover the sequence of each computation step and provide an extractor with a universal strategy.

To address this obstacle, we leverage random oracles. Specifically, we design our PoSt s.t., for an admissible PoSt prover, all PoR challenges except the first one must be generated from the random oracle, while all other PoR proofs except the last one must be queries to the random oracle. Since the PoSt prover is evaluated, and the extractor simulates the random oracle, PoR challenges and responses can be seen by the extractor. It becomes possible for the extractor to invoke a PoR extractor and recover the data by controlling the random oracle. Our extraction strategy may be of independent interest.

1.2 Related works

Ateniese et al. introduced Provable Data Possession (PDP) [8] to allow a cloud provider to prove to its clients that their data is intact and available. Proof of Retrievablity (PoR) was initially proposed by Juels and Kaliskiand [31] and improved in a series of subsequent works [44, 18, 24, 47, 45, 20, 4]. PoR requires the existence of an extractor to completely recover the stored data. PDP/PoR can be extended with various advanced features, such as supporting dynamic updates [20, 45], multiple servers [21, 18, 36], or replication [4, 37, 28, 23]. For better efficiency, it's possible to outsource PoR's verification workload to a third party auditor [5]. However, the auditor must continuously challenge the storage server and "compress" and forward the responses to the data owner. Moreover, the construction in [5] requires a trusted randomness beacon to generate PoR challenges periodically.

A primitive named Proof of Space-Time has recently been proposed by Moran and Orlov [33]. Their notion is distinct from the one introduced in the Filecoin whitepaper and that we consider in this paper. In their paper, "space-time" is meant to capture the use of space as a resource over time but does not consider the availability of content. In this respect, their concept can be viewed as an extension of Proof of Space [6, 27]. Another similar notion is sustained-memory complexity [3], which requires that any algorithm evaluating a specific function must use a large amount of memory for many steps.

2 Preliminary

In this section, we introduce several preliminary notions.

2.1 Interactive Turing machine.

An interactive Turing machine (ITM) is to model interactive algorithms used in real-life computing systems. It was initially used by Goldwasser, Micali, and Rackoff [30] to model interactive proof systems. An ITM has an input tape, an output tape, a randomness tape, and k working tapes, and it changes its state step by step following the instruction described by a transition function.

We have the following definition of an ITM.

Definition 1. *An interactive Turing machine (ITM) with k work tapes is a* 5*-tuple*

$$\mathcal{T} = (\Sigma, \Omega, Q, q_{init}, F, \delta),$$

where

- Σ *is a non-empty alphabet,*
- Q *is a set of states,*
- q_{init} *is the initiate state,*
- F *is the state of finial states,*
- *The transformation function*

$$\delta : Q \times \Sigma \times \Sigma \times \Sigma^k \longrightarrow$$
$$Q \times \{none, right\} \times \{none, right\} \times \left(\Sigma^k, \{left, none, right\}^k\right) \times (\{\Sigma \cup \perp\}, \{none, left\}).$$

At each step, the ITM will read one symbol from the input tape, one symbol from the random tape and k symbols from the work tapes. Then based on the current state and the transformation function δ, it will decide

- *which state to change to,*
- *to move the header on the reading tape to right or stay,*
- *to move the header on the random tape to right or stay,*
- *to print k symbols on the work tapes,*
- *to move the headers on work tapes to right or left or stay,*
- *to print one symbol on the writing tape or give up to output,*
- *to move the header on the writing tape to left or stay.*

It is convenient to use a transition function based on configurations to describe the execution procedure of an ITM. A configuration consists of a state, the contents of the tapes and the positions of the tape headers, denoted as $(q, v_1, \ldots, v_k, i_1, \ldots, i_k)$, for a state $q \in Q$, strings on one of work tapes $v_j \in \Sigma^*$ and integers $1 \leq i_j \leq |v_j + 1|$ for every $1 \leq j \leq k$. Let $\mathbf{Q}$ be the set of configurations. The initial configuration $\mathbf{q_{init}}$ is the configuration consisting of the initial state and k empty tapes, with the tape heads at the first position. So the transition function can be written as

$$\delta : \mathbf{Q} \times \Sigma \to \mathbf{Q} \times \{none, right\} \times \{\Sigma \cup \perp\}.$$

Consequently, the running of an ITM is sequentially changing the configurations, according to the transition function and the symbols on the input tape. More importantly, given one configuration at a specific time and the transition function, anyone can run the ITM from that point on.

2.2 Proof of retrievability.

Proof of retrievability [31, 44] is a proof-of-storage scheme which provides strong retrievability guarantees. The soundness of PoR requires that if a server can pass an audit then a special extractor algorithm, interacting with the server, must be able (w.h.p.) to extract the file. Our PoR syntax is adapted from [44] by Shacham and Waters, except that we specify the interaction between the prover and the verifier as a challenge-and-response procedure. Formally, a proof of retrievability scheme defines four algorithms, $\mathsf{PoR.Kg}$, $\mathsf{PoR.Store}$, $\mathsf{PoR}.\mathcal{V}$ and $\mathsf{PoR}.\mathcal{P}$:

- $\mathsf{PoR.Kg}$: Generate a public-private keypair (pk, sk).
- $\mathsf{PoR.Store}(sk, D)$: Taking as input a secret key sk and a file $D \in \{0,1\}^*$, the "setup" algorithm encodes D into D^* as the file to be stored, and generates a public tag tg for further proof and verification.
- $\mathsf{PoR}.\mathcal{V}$: The verification algorithm consists of two parts: 1). $\mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}$, which generates a challenge c, and 2). $\mathsf{PoR}.\mathcal{V}_{\mathsf{valid}}$, which verifies the response p from the prover $\mathcal{P}$ corresponding to the challenge c. Specifically, $\mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}(pk, sk, tg)$ takes the public key pk, the secret key sk and the tag tg as inputs, and generates a challenge c. $\mathsf{PoR}.\mathcal{V}_{\mathsf{valid}}(pk, sk, tg, c, p)$ takes the public key pk, the secret key sk, the tag tg, the challenge c, the corresponding proof p as inputs, and outputs a bit b which is either 1 or 0 to indicate whether the verifier accepts or not.

– $\mathsf{PoR}.\mathcal{P}(pk, tg, D^*, c)$: The proving algorithm takes as input the public key pk, the file tag tg output by $\mathsf{PoR}.\mathsf{Store}$, the encoded file D^* and the challenge c, and outputs a proof p after computation.

Publicly verifiable. If the verification algorithm $\mathsf{PoR}.\mathcal{V}$ does not need to take the secret key sk as input, we call this PoR scheme publicly verifiable.

Stateful/stateless PoR. A PoR scheme can be either stateful [31] or stateless [44]. A PoR scheme is stateful if the number of audit interactions between the prover and verifier is bounded, and the verifier has to maintain a state to record the number of interactions. While a PoR scheme is stateless if the verifier does not need to maintain a state, and can invoke the audit procedure an unlimited (polynomial) number of times.

Correctness and soundness. A PoR scheme should satisfy both correctness and soundness. Correctness requires that the verification algorithm always accepts the proof when interacting with an honest prover. Soundness aims to model that any party who can convince the verifier must be storing that file. The formal soundness definition of PoR follows the classical notion of proof of knowledge [44]. Mainly, soundness requires that for any ITM $\mathcal{P}'$ generated by the adversary that implements a legitimate prover in the proof-of-retrievability protocol, there is an extractor algorithm $\mathsf{Extr}(pk, sk, tg, \mathcal{P}')$ taking as input the public and private keys, the file tag tg, and the description of the ITM $\mathcal{P}'$, that outputs the file $D \in \{0,1\}^*$. Note that Extr is given non-black-box access to $\mathcal{P}'$ and can, in particular, rewind it. The logic behind this extractability definition is that the best way to guarantee the prover possesses the data is to recover it via interacting with the prover.

Unpredictability. To facilitate our PoSt construction, we define a special property for the challenge response style of PoR, named *unpredictability*. It ensures that the prover cannot guess a valid response before he sees the corresponding challenge. Formally, we have the following definition.

Definition 2. *A challenge-response style PoR scheme is unpredictable if for any P.P.T adversary $\mathcal{A}$, the following holds,*

$$\Pr\left[p \leftarrow_{\$} \mathcal{A}(pk, tg, D^*) \wedge 1 \leftarrow \mathcal{V}_{\mathsf{valid}}(pk, sk, tg, c, p) \quad \middle| \, c \leftarrow_{\$} \mathcal{V}_{\mathsf{cha}}\right] < \mathsf{negl}(\lambda).$$

where $\mathsf{PoR}.\mathcal{V} := (\mathcal{V}_{\mathsf{cha}}, \mathcal{V}_{\mathsf{valid}})$, and λ is the security parameter.

Note that unpredictability is not provided by default[5], but it is achieved by most existing PoR schemes, e.g., compact PoR [44], since the prover's response has enough high entropy.

3 Delay Function

A delay function is a function $F : \mathcal{X} \to \mathcal{Y}$ that, even when using multiple processors and parallelism, cannot be evaluated in less than a prescribed time [14]; while, on the other hand, there exists an algorithm so that honest evaluators can terminate the computation in a similar amount of time. Here we introduce two variants of the delay function. One is the *verifiable* delay function (VDF) [14], which enables the evaluator to generate a succinct proof to show the correctness of the result. The other is the *trapdoor* delay function (TDF) [48], which enables the holder of the secret trapdoor to evaluate the function without delay. The formal definitions are given next.

3.1 Verifiable delay function

A VDF is a scheme consisting of the following three algorithms [15, 48]:

– $\mathsf{VDF}.\mathsf{Setup}(\lambda, s)$ is a randomized algorithm that given as input the security parameter λ and a delay parameter s (measured by Turing machine steps), generates the public parameters pp. The input and output spaces, $\mathcal{X}$ and $\mathcal{Y}$, are determined by pp. For meaningful security, the delay parameter s is restricted to be sub-exponentially sized in λ.

[5] A counterexample is that the prover returns all the data independently of the challenge.

– $\mathsf{VDF.Eval}(pp, x)$ is a randomized algorithm that given as input the public parameters pp and $x \in \mathcal{X}$, outputs the answer $y \in \mathcal{Y}$ and a proof π.
– $\mathsf{VDF.Verify}(pp, x, y, \pi)$ is a deterministic algorithm that given as input the public parameters pp, $x \in \mathcal{X}$, $y \in \mathcal{Y}$, and the proof π, emits one bit 1 or 0 to denote an acceptance or a rejection.

A VDF must satisfy the following three properties [15]:

- **δ-evaluation time:** For all pp generated by $\mathsf{VDF.Setup}(\lambda, s)$ and all $x \in \mathcal{X}$, the algorithm $\mathsf{VDF.Eval}(pp, x)$ must run in steps $(1 + \delta)s$ with $\mathsf{poly}(\log(s), \lambda)$ processors.[6]
- **Sequentiality:** A parallel algorithm $\mathcal{A}$, using at most $\mathsf{poly}(\lambda)$ processors, that runs in sequential steps less than s cannot compute the function. Specifically, for a random $x \in \mathcal{X}$ and pp output by $\mathsf{VDF.Setup}(\lambda, s)$, if $(y, \pi) \leftarrow \mathsf{VDF.Eval}(pp, x)$ then $\Pr[\mathcal{A}(pp, x) = y]$ is negligible.
- **Uniqueness**: For an input $x \in \mathcal{X}$, exactly one $y \in \mathcal{Y}$ will be accepted by $\mathsf{VDF.Verify}$. Specifically, let $\mathcal{A}$ be an efficient algorithm that given pp as input, outputs (x, y, π) such that $\mathsf{VDF.Verify}(pp, x, y, \pi) = accept$. Then $\Pr[\mathsf{VDF.Eval}(pp, x)] \neq y]$ is negligible.

3.2 Trapdoor delay function

A TDF $F : \mathcal{X} \to \mathcal{Y}$ is a scheme consisting of the following three algorithms [48]:

– $\mathsf{TDF.Setup}$ (λ, s) is a randomized algorithm that takes as input a security parameter λ and a delay parameter s (measured by Turing machine steps), and outputs public parameters pp and a trapdoor tr. The delay parameter s is sub-exponential in λ.
– $\mathsf{TDF.Eval}(pp, x)$ takes as input $x \in \mathcal{X}$ and outputs a $y \in \mathcal{Y}$.
– $\mathsf{TDF.TrapEval}(pp, tr, x)$ takes as input x and a trapdoor tr, outputs a $y \in \mathcal{Y}$.

TDF must satisfy **δ-evaluation time** and **sequentiality** as in the case of standard VDF. Similarly, we assume $0 < \delta \ll 1$ for TDF. In fact, the gap between the honest evaluator and the malicious one that is characterized by δ could be even smaller than that in VDF, because no proof needs to be generated. Besides, the following two unique requirements must be satisfied by TDF:

- **Trapdoor efficiency:** $\mathsf{TDF.TrapEval}$ must run in total steps polynomial in $O(\log s)$ and λ. Therefore $\mathsf{TDF.TrapEval}$ is much faster than $\mathsf{TDF.Eval}$.
- **Correctness:** $\mathsf{TDF.Eval}$ and $\mathsf{TDF.TrapEval}$ will produce the same result on the same input.

The TDF can be easily instantiated via the RSA trapdoor as following [48]:

– $\mathsf{TDF.Setup}(\lambda, s)$: Output two objects:
  • A finite abelian group $\mathbb{G}$ of unknown order
  • An efficiently computable hash function $H : \mathcal{X} \leftarrow \mathbb{G}$ that we model as a random oracle.

  We set the public parameters pp to be $pp := (\mathbb{G}, H, s)$ and the trapdoor tp to be the real order d of $\mathbb{G}$.
– $\mathsf{TDF.Eval}(pp, x)$: compute $y \leftarrow H(x)^{2^s} \in \mathbb{G}$ by computing T squaring in $\mathbb{G}$ starting with $H(x)$, and output y
– $\mathsf{TDF.TrapEval}$ (pp, tr, x): Let d be the order of the group and $2^s \bmod d = r'$, we just need to compute $y = H(x)^{r'}$.

For an implementation, one can choose $\mathbb{G}$ as the RSA group, so the trapdoor $d = \phi(N)$ where N is the RSA modulus and ϕ is the Euler's function.

[6] It has recently been shown in [25, 48] that one can convert a VDF into a tight one which can be evaluated in sequential steps $s + O(1)$ with an honest prover using $O(log(s))$ processors and space. Therefore, without loss of generality, we assume $0 < \delta \ll 1$ in the following sections.

4 Formalizing Proof of Storage-Time

Now we are ready to give a formal definition of PoSt. Recall that PoSt is a protocol that enables the verifier to audit that the data is continuously available at the server for a specific range of time. The syntax of PoSt is similar to the challenge-response style of PDP/PoR in section 2.2, except that time parameters must now be considered. How to measure time is a tricky question, and indeed, time is hard to capture by algorithms. Instead, we consider using the number of unit steps of a Turing machine (which can be seen as a mathematical abstraction of CPU clock cycles) as a measure of time similar to the time-lock puzzles [42] and the VDF [14]. In the following, the time parameters are all represented as the number of steps of the Turing machine.

The ideal version of continuous data availability is a continuous notion that makes it challenging to instantiate. A discretized approximation seems to be inevitable, and this can be accomplished by choosing an audit frequency parameter t. Hence, the large time range T can be arbitrarily divided into time segments of length t, and the prover must provide a valid proof at least once in every time slot. Obviously, a smaller t would result in a better availability guarantee.

Specifically, the key generation phase of PoSt takes as input the audit frequency parameter t, and a deposit time T which expresses the time that the data file is supposed to be stored at the server. Besides, the verifier needs to keep a timer for checking whether the final proof is received on time. For simplicity, we do not consider the communication latency. Formally, a PoSt scheme consists of the following four algorithms.

- $\mathsf{PoSt.Kg}(\lambda, t, T)$: Given the security parameter λ, the audit frequency parameter t and the data storage time T, this randomized algorithm generates a public-private key pair (pk, sk).
- $\mathsf{PoSt.Store}(sk, D)$: The file-storing algorithm takes as input a secret key sk and the content $D \in \{0,1\}^*$, encodes D into D^* as the file to be stored, and computes a tag tg for further proof and verification.
- $\mathsf{PoSt}.\mathcal{V}$ is the verification algorithm which has two subroutines $\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}$ and $\mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}}$ for the challenge generation and the response validation, respectively. $\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}(pk, sk, tg)$ takes as input the public key pk, the secret key sk and the tag tg, and generates a challenge c as well as setting a public timer to be 0. $\mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}}(pk, sk, tg, c, p, timer)$ takes as input the public key pk, the secret key sk, the tag tg, the challenge c, the corresponding response p and the time of the timer for receiving the response p, and outputs a bit b to indicate "reject" or "accept".
- $\mathsf{PoSt}.\mathcal{P}(pk, tg, D^*, c)$: The randomized proving algorithm takes as input the public key pk, the file tag tg output by St, the decoded file D^* and the challenge, outputs a response p after a period of computation and sends it back to the verifier immediately after the computation is finished.[7]

A PoSt scheme may possess other advanced features.

Compactnesss. A PoSt scheme is compact, if the cost of the verification algorithm $\mathsf{PoSt}.\mathcal{V} = (\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}, \mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}})$ is independent of the storage time T and the size of the file.

Public verifiability. Similar to PDP/PoR, if the subroutines $\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}$ and $\mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}}$ of the verify algorithm do not need the secret key sk as input, we call this scheme *publicly verifiable*. It implies that continuous data possession can be verified by any third party, not only by the data owner.

Public validation. Consider the case where only $\mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}}$ does not require the secret key as input, while $\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}$ still needs it. In this case, the response p can be publicly validated, given the challenge c. We call these PoSt schemes *publicly validatable*. Unlike publicly verifiable schemes, $\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}$ may still take sk as input so that challenges cannot be generated publicly. Therefore, the timer for verification should be initiated by the data owner but can be seen by everyone. Although not as flexible as publicly verifiable schemes, validatable PoSt schemes are sufficient for many useful applications, including those related to decentralized storage markets, as advocated by Filecoin.

[7] In practice, $\mathsf{PoSt}.\mathcal{P}$ is supposed to send back the final response after the storage period ends; otherwise it is impossible to detect malicious attempts such as discarding the stored data at the last moment. Therefore, either the computation of $\mathsf{PoSt}.\mathcal{P}$ takes longer time than T, or an intentional delay is included in the algorithm.

[8] In particular, data owners can outsource their data to any party in the network and publish a privately-generated challenge together with the parameters of the smart contract (the public key pk, the tag tg, and the initial time t_1). Any party that stores the data can submit a response p in time t_2, and the contract can publicly evaluate $\mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}}$ with pk, tg, p and $t_2 - t_1$ as input. The storage provider earns his reward if the output bit b equals to 1.

Stateful/Stateless PoSt. A PoSt scheme is *stateful*, if it supports a limited number of audits after the setup procedure and a state must be maintained. The state is updated after a challenge is used, i.e., $\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}(pk, sk, tg, state) \rightarrow (c, timer, state)$. The $\mathsf{PoSt}.\mathsf{Store}$ algorithm is parameterized by an integer ℓ, which indicates the maximum number of interactions. The variable *state* records the times that the server has been queried. In practice, a new challenge cannot be generated when the bound is reached unless the data owner retrieves all the data and relaunches the store procedure. On the contrary, the audit interaction for a *stateless* PoSt can be invoked an unbounded (polynomial) number of times.

Generally, a PoSt scheme should satisfy both properties of *Correctness* and *Soundness*.

4.1 Correctness

Correctness requires that the verification algorithm accepts the proof when interacting with a valid prover.

Definition 3. *A stateless PoSt scheme is correct if for all keypairs (pk, sk) output by $\mathsf{PoSt.Kg}(\lambda, t, T)$, for all files $D \in \{0,1\}^*$, and for all (D^*, tg) output by $\mathsf{PoSt.Store}(sk, D)$, the verification algorithm accepts the proof when interacting with a valid prover. Specifically, if p is the response generated by $\mathsf{PoSt}.\mathcal{P}(pk, tg, D^*, c)$ on the challenge c generated by $\mathsf{PoSt}.\mathcal{V}_{cha}(pk, sk, tg)$ and sent back to the verifier immediately after the proof computation is finished, $\mathsf{PoSt}.\mathcal{V}_{valid}(pk, sk, tg, c, p, timer)$ always outputs 1.*

For the stateful PoSt, the correctness definition is the same, except the state is involved in $\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}$.

4.2 Soundness

As illustrated in the introduction, providing a suitable and rigorous definition for soundness is a challenging task. A natural choice is to upgrade the PoR soundness, which requires an extractor algorithm to extract the data while interacting with a legitimate prover. This follows the classical definition of proof of knowledge. It is not hard to imagine, in an ideal version of PoSt, to have an extractor that extracts the data possessed by a legitimate prover at any specific point within the time range T. Our strategy is to characterize the prover algorithm as an Interactive Turing Machine (ITM), which is executed to generate a proof after receiving a challenge. We intend to mimic the situation where someone runs a computer but ends at a specific time point, "freezes" its memory, and then checks the data preserved on the machine. Intuitively, the data on a computer should be either preserved in the memory (characterized as configurations) or hardcoded in the program (characterized as the transition function). Therefore, for the notion of soundness, the data should be extracted from the configuration corresponding to any specific time and the transition function. To facilitate the construction, we provide a more general definition for the above-idealized version: instead of selecting a configuration of one step, we allow the extractor to choose a bunch of configurations that correspond to a time slot with length t.

In general, the soundness experiment of PoSt consists of two procedures as in PDP/PoR, i.e., the setup game and the extraction algorithm. The setup game lets the adversary generate a prover algorithm $\mathcal{P}'$. In the extraction algorithm, the extractor recovers the data from the generated algorithm $\mathcal{P}'$. The setup game of PoSt is similar to that of PDP/PoR, while the extraction procedure needs to be defined specifically.

[8] Ideally, they would also demand the stronger public verifiability property so that PoSt could be integrated into the mining procedure.

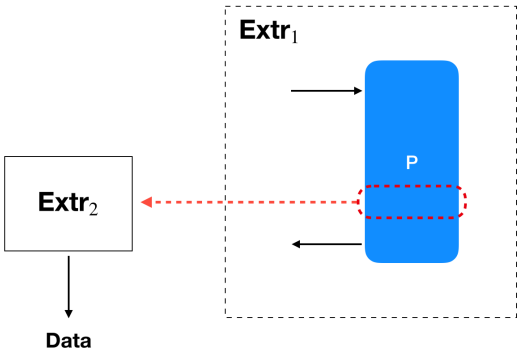


Fig. 2. PoSt soundness model.

Setup Game The first procedure is the following *setup* game between an adversary $\mathcal{A}$ and an environment, which simulates the procedure that the adversary generates a malicious prover $\mathcal{P}'$ by getting arbitrary storage data from an honest data owner and then freely interacts with the verifier.

Step 1. The environment generates a keypair (pk, sk) by running $\mathsf{PoSt.Kg}$, and provides pk to $\mathcal{A}$
Step 2. The adversary now interacts with the environment. It can make queries to a *store* oracle, providing, for each query, some file D. The environment computes (D^*, tg) by invoking $\mathsf{PoSt.Store}(sk, D)$ and gives both D^* and tg back to $\mathcal{A}$. If the PoSt is stateful, the environment initiates and maintains a verifier's state for each storage query.
Step 3. For any D on which it previously made a *store* query, the adversary undertakes executions of the PoSt challenge-response interaction, by specifying the corresponding tag tg. In these protocol executions, the environment plays the part of the verifier, and the adversary plays the part of the prover. Specifically, the environment maintains a timer for each interaction. When a protocol execution completes, the adversary is provided with the output of $\mathcal{V}$. These protocol executions can be arbitrarily interleaved with each other and with the *store* queries described in Step 2. For the stateful PoSt, the environment also updates the state as a verifier, so the adversary can only interact with the verifier a limited number of times for a specific storage.
Step 4. Finally, the adversary outputs a challenge tag tg returned from some *store* query, and the description of a prover $\mathcal{P}'$, which is an ITM defined before.

Specifically, the prover $\mathcal{P}'$ for a file D and storage time T is ϵ-admissible for a stateless PoSt if it convincingly answers an ϵ fraction of challenges, i.e., $\Pr[\langle \mathsf{PoSt}.\mathcal{V}(\cdot), \mathcal{P}' \rangle = 1] \geq \epsilon$. Similarly, for the stateful scheme we require that $\Pr[\langle \mathsf{PoSt}.\mathcal{V}(\cdot, state), \mathcal{P}' \rangle = 1] \geq \epsilon$ for any $state$ from 1 to l. The probability is over the coins of the verifier and the prover.

Extraction The second procedure is to allow the extractor to extract the data from the prover $\mathcal{P}'$. As discussed above, we provide a generalized notion of "continuous extractability", which aims to capture that if one chooses any time period with length t during the storage time, the data is available for at least one time point in this period. Intuitively, if the interval between two data audits is larger than t, the data may not be available for this interval, and hence cannot be extracted from the corresponding bunch of configurations and the transition function. So t actually characterizes the largest interval between two data audits, and it represents the checking frequency parameter.

Specifically, we design an extract experiment which has two procedures. To guarantee that the extracted data is indeed "on the machine" during that chosen time slot, we run two procedures. The first procedure, named Extr_1, is to honestly run the algorithm $\mathcal{P}'$ as an ITM, to get the sequence of the configurations, and to cut out a bundle of sequential ITM configurations which correspond

to a length of time t. The second procedure, named Extr_2, can use this bundle of sequential ITM configurations, along with the transition function of $\mathcal{P}'$, to extract the data.

In our definition, the behavior of Extr_1 is fixed, but the strategy of Extr_2 is arbitrary. Moreover, the execution time of $\mathcal{P}'$ must be longer than the deposit time T. Particularly, when given the transition function and a bunch of configurations corresponding to one time slot, Extr_2 may rewind a portion of the computations of $\mathcal{P}'$ by himself. The sketch of the experiment can be seen in Fig. 2.

Definition 4 (Soundness). *A PoSt scheme is ϵ-sound if for every adversary $\mathcal{A}$ which plays the setup game and outputs an ϵ-admissible prover $\mathcal{P}'$ for a file D as an ITM with k tapes, there is an extractor Extr which has two subroutine ITMs, Extr_1 and Extr_2, that can perform the following tasks.*

- Extr_1 *is an ITM with $k+1$ working tapes. Extr_1's input is the description of $\mathcal{P}'$. He devotes k of his working tapes to simulate all the tapes of $\mathcal{P}'$. Extr_1 first writes an arbitrary challenge on the simulated input tape of $\mathcal{P}'$, then simulates every step of $\mathcal{P}'$ following the instructions of $\mathcal{P}'$'s transition function. After each step of $\mathcal{P}'$, Extr_1 records the current configurations of $\mathcal{P}'$ on his extra tape before starting to simulate the next step of $\mathcal{P}'$. After finishing all of $\mathcal{P}'$ steps, Extr_1 will randomly pick t successive configurations of $\mathcal{P}'$ to write on his output tape together with the description of the transition function of $\mathcal{P}'$.*
- *The input of Extr_2 is directly obtained from the output tape of Extr_1. Extr_2 must recover D purely from the information s returned by Extr_1 except a negligible probability, no matter which t successive configurations of $\mathcal{P}'$ is chosen by Extr_1. Hence the probability*

$$\Pr\left(D \leftarrow_{\$} \mathit{Extr}_2(s) | s \leftarrow_{\$} \mathit{Extr}_1(\mathcal{P}')\right) > 1 - \mathit{negl}(\lambda)$$

for the randomnesses of both Extr_1 and Extr_2.

5 Constructions for PoSt

In this section, we provide two PoSt constructions which are proven secure in the above model. Each of them has its own advantages.

The first construction follows the structure of the ad-hoc proposal from the Filecoin white paper [40] but can be proven secure. Intuitively, it uses a VDF and random oracles to force the prover to generate PoR proofs sequentially. This scheme is stateless, which means that the *prove* procedure can be launched an unlimited number of times after *storing*. Also, this scheme is publicly verifiable; hence a third party can verify the continuous availability of data by interacting with the server. The drawback of this scheme is that the size of the final proof is linear with respect to the storage time. Given that the verification time is also linear, we consider this just a warm-up construction.

The next construction is our main one which achieves compact proofs, so its communication size is independent of the default time. Particularly, its verification algorithm is extremely efficient. Nevertheless, this construction is stateful, i.e., after a limited number of PoSt interactions, no more proofs can be generated until rerunning the storing phase. Moreover, this scheme provides public validation, but it is not publicly verifiable. This means that the verifier needs a secret key for the generation of the challenge; however, the verification of the PoSt proof does not.

5.1 Basic PoSt Scheme

Next, we describe our first construction called the *basic PoSt*. Our main building blocks are a stateless unpredictable PoR scheme (defined in Subsection 2.2) and a VDF scheme. The main intuition is to force the PoSt prover to sequentially generate one PoR proof every once in a while during the entire storage period. To achieve this goal, we let the challenges of the next PoR be derived from the output of VDF, whose input is the previous PoR proof (see the right side of Fig. 1). Therefore, a malicious prover cannot generate all the PoR proofs at once at the beginning of the storage period, because each PoR procedure is connected to the next one in series by a delay function. Also, the prover cannot discard the data first and wait until the last moment to retrieve it and generate all the PoR proofs at once, otherwise he will risk not sending the final response in time due to the delay function.

Specifically, the following primitives will be used in the construction. A stateless publicly-verifiable PoR scheme with *unpredictability* consists of a tuple of algorithms

$$(\mathsf{PoR.Kg}, \mathsf{PoR.Store}, \mathsf{PoR}.\mathcal{P}, \mathsf{PoR}.\mathcal{V})$$

where $\mathsf{PoR}.\mathcal{V} = (\mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}, \mathsf{PoR}.\mathcal{V}_{\mathsf{valid}})$. $\mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}$ and $\mathsf{PoR}.\mathcal{V}_{\mathsf{valid}}$ denote the verifier's algorithms for generating the challenge and verifying the proof, respectively. $\mathcal{H}$ and $\mathcal{G}$ are hash functions that can be viewed as random oracles. $(\mathsf{VDF.Setup}, \mathsf{VDF.Eval}, \mathsf{VDF.Verify})$ are the algorithms for a verifiable delay function with δ-evaluation time for some constant δ. D denotes the storage file. The PoSt scheme needs a global timer which is initiated by the data owner but can be seen by everyone.

The PoSt is parameterized by the storage time T and checking frequency t. Without loss of generality, T and t are measured by the number of steps of a Turing machine (to mimic the CPU clock). Note that since the adversary may evaluate the delay function slightly faster than the honest prover, we require them to check the data more frequently than the requested frequency t. Based on T and t, PoSt scheme will choose the VDF delay time $t' \leq t - 2\delta T$. Precisely, we set the delay parameter of VDF as t'. Hence, the sequential steps for evaluating VDF should be larger than t' for any parallel adversary with $\mathsf{poly}(\lambda)$ processors; on the other hand, the honest server can finish the VDF evaluation within time $(1+\delta)t'$ (see Section 6.2 for more detailed discussions about the parameter setting). Moreover, we assume that the time cost of generating one PoR proof and evaluating one hash function is much smaller than t (thus, it can essentially be ignored), and our construction guarantees that the largest time interval between two PoRs is less than $t' + 2\delta T \leq t$.

The PoSt works as follows:

- $\mathsf{PoSt.Kg}(\lambda, t, T)$: Use $\mathsf{PoR.Kg}$ to generate a PoR public-private key pair $(\mathsf{PoR}.pk, \mathsf{PoR}.sk)$. t' is the largest number of ITM steps such that $t' \leq t - 2\delta T$ and $k = T/t'$ is an integer. Use $\mathsf{VDF.Setup}\,(\lambda, t')$ to generate the public parameter $\mathsf{VDF}.pp$ for VDF where the time cost is at least t'. The public key pk of PoSt is $(\mathsf{PoR}.pk, \mathsf{VDF}.pp, T, k)$ and the secret key sk is $\mathsf{PoR}.sk$.
- $\mathsf{PoSt.Store}(pk, sk, D)$: Take a secret key sk and a file $D \in \{0,1\}^*$ as input. Use $\mathsf{PoR.Store}$ to process D and output D^* and a tag $tg = \mathsf{PoR}.tg$.
- $\mathsf{PoSt}\ .\mathcal{V} = (\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}, \mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}})$:
  - $\mathsf{PoSt}.\mathcal{V}_{\mathsf{cha}}(pk, tg)$: Use $\mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}$ to generate a challenge c_0 and set its timer to 0.
  - $\mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}}(pk, tg, c_0, p)$: When the PoSt proof p is received from the prover, $\mathsf{PoSt}.\mathcal{V}_{\mathsf{valid}}$ first check the current timer T'. If the timer T' is smaller than T or larger than $(1+\delta)T$, output $reject$, otherwise run Algorithm 2 with input p, tag tg and processed data D^* and release its output. Intuitively, the verifier needs to check all hash evaluations, all PoR proofs, and all VDF evaluations.
- $\mathsf{PoSt}.\mathcal{P}(pk, tg, D^*, c_0)$: Defined as Algorithm 1. Intuitively, the prover sequentially computes PoR instances where the next PoR challenge is generated from the previous PoR proofs via hash functions and VDF.

Algorithm 1 The PoSt prove algorithm $\mathsf{PoSt}.\mathcal{P}$

Require: The initial challenge c_0, the stored data D^*, the public key $pk = (\mathsf{POR}.pk, \mathsf{VDF}.pp, k, T)$ and the tag $tg = \mathsf{PoR}.tg$

Ensure: The PoSt proof p

1: **for** $i = 0$ to $k-1$ **do**
2: $\quad v_i \leftarrow \mathsf{PoR}.\mathcal{P}\,(c_i, D^*, \mathsf{PoR}.pk, \mathsf{PoR}.tg)$ *// Generate a PoR proof.*
3: $\quad u_i = \mathcal{G}(v_i)$
4: $\quad (d_i, \pi_i) \leftarrow \mathsf{VDF.Eval}(u_i, \mathsf{VDF}.pp)$ *// Compute VDF while generate its proof.*
5: $\quad c_{i+1} = \mathcal{H}(d_i)$
6: $v_k \leftarrow \mathsf{PoR}.\mathcal{P}(c_k, D^*, \mathsf{PoR}.pk, \mathsf{PoR}.tg)$
7: $p = \left(\{c_i, v_i\}_{i=0}^{k}, \{u_i, \pi_i, d_i\}_{i=0}^{k-1}\right)$
8: **return** p

Algorithm 2 PoSt Verification Algorithm

Require: The PoSt proof p, the public key $pk = (\mathsf{POR}.pk, \mathsf{VDF}.pp)$, and the tag $tg = (\mathsf{PoR}.tg, k, T)$
Ensure: The verification result b
parse p as $\left(\{c_i, v_i\}_{i=0}^{k}, \{u_i, \pi_i, d_i\}_{i=0}^{k-1}\right)$
for $i = 0$ to $k-1$ **do**
  if $u_i \neq \mathcal{G}(v_i)$ **then return** *reject*
  if $c_i \neq \mathcal{H}(d_i)$ **then return** *reject*
  if $0 \leftarrow \mathsf{PoR}.\mathcal{V}_{\mathsf{valid}}(\mathsf{PoR}.pk, \mathsf{PoR}.tg, c_i, v_i)$ **then return** *reject*
  if $0 \leftarrow \mathsf{VDF}.\mathsf{Verify}(\mathsf{VDF}.pp, d_i, u_i, \pi_i)$ **then return** *reject*
if $0 \leftarrow \mathsf{PoR}.\mathcal{V}_{\mathsf{valid}}(\mathsf{PoR}.pk, \mathsf{PoR}.tg, c_k, v_k)$ **then**
  return *reject*
else
  return *accept*

Correctness. Note that the final PoSt proof procedure includes k VDF evaluations where $k = T/t'$. Since we assume that the time spent evaluating one VDF (with δ-evaluation time) is shorter than $(1+\delta)t'$, and the time cost for PoRs and evaluating hash is comparatively negligible, the total time for an honest prover to generate a PoSt proof is less than $(1+\delta)T$. Hence the correctness of our PoSt directly follows from those of the PoR and the VDF schemes.

Soundness. Our goal is to prove that the largest time interval between two PoRs is less than t. Therefore, for an admissible prover, any successive configurations of any time epoch with length t must contain at least one PoR. Ideally, one can use the PoR extractor to recover the data from the partial configurations and the transition function. However, one problem is that since the strategy of a malicious prover cannot be predicted, it is hard to let the extractor access each PoR's challenge and response. To solve this problem, our soundness proof fully exploits the unpredictability of the random oracle. Specifically, since an admissible prover must inevitably query the random oracle, the challenge and response for each PoR can be located via querying the random oracle $\mathcal{G}$ and $\mathcal{H}$ respectively, hence we can extract the data via the PoR extractor.

Theorem 1. *PoR is a stateless PoR scheme with ϵ-soundness and unpredictability. VDF is a VDF scheme with δ-evaluation time. The time cost of PoR and evaluating a hash function is negligible w.r.t. t. The time cost of s_0 sequential steps on the server processor is t'. If $t' + 2\delta T < t$, the PoSt scheme is stateless and has ϵ-soundness.*

Proof. Assume that an adversary $\mathcal{A}$ outputs a cheating prover $\mathsf{PoSt}.\mathcal{P}'$ in the setup game, which can return a valid proof $p = \left(\{c_i, v_i\}_{i=0}^{k}, \{u_i, \pi_i, d_i\}_{i=0}^{k-1}\right)$ with probability ϵ. To show the soundness of the scheme, we need to construct the extractor $\mathsf{PoSt}.\mathsf{Extr}_2$ which can recover the data D from the successive configurations of a t length time epoch and transit function returned by $\mathsf{PoSt}.\mathsf{Extr}_1$, no matter which epoch is chosen by $\mathsf{PoSt}.\mathsf{Extr}_1$. Generally, our proof consists of two steps. In the first step, we prove that the prover will execute one PoR in the epoch randomly chosen by $\mathsf{PoSt}.\mathsf{Extr}_1$. In the second step, we will invoke the PoR extractor to recover the data from the configurations the epoch and the transition function.

For the first step, let T_0 and T_k are the starting and ending time points for running $\mathsf{PoSt}.\mathcal{P}'$. For i from 1 to $k-1$, we set each time point T_{i+1} to be the first time when $\mathsf{PoSt}.\mathcal{P}'$ queries the random oracle $\mathcal{H}$ on d_i. Similarly, we set each time point R_i as the first time when $\mathsf{PoSt}.\mathcal{P}'$ queries the random oracle $\mathcal{G}$ on v_i. Then we will prove that:

Claim 1) T_i must precede T_{i+1},
Claim 2) the length of each time slot $[T_i, T_{i+1})$ is longer than t',
Claim 3) the length of each time slot $[T_i, T_{i+1})$ is shorter than $t' + \delta T$,
Claim 4) each R_i belongs to the time slot $[T_i, T_{i+1})$ and the time slot $[T_i, R_i)$ is shorter than δT.

If above claims are all proved, the random time epoch with length $t > t' + 2\delta T$ chosen by $\mathsf{PoSt}.\mathsf{Extr}_1$ must contain at least one interval $[T_i, R_i)$ for some i. This is because $T_1, \ldots, T_{k-1}$ divides the whole execution time of $\mathsf{PoR}.\mathsf{Extr}$ into k slots whose lengths are all shorter than t. So any epoch with length t must contain some T_i. Since $t > t' + 2\delta T$, the epoch must contain either T_{i-1} or R_i,

otherwise the length of the interval $[T_{i-1}, R_i)$ is longer than t. So either the interval $[T_{i-1}, R_{i-1})$ or $[T_i, R_i)$ is contained in this epoch.

Next we prove the above claims one by one.

For Claim 1), we show that each d_{i-1} must be firstly queried to the random oracle $\mathcal{H}$ before d_i. We prove it by contradiction. If not, $\mathsf{PoSt}.\mathcal{P}'$ must be able to either generate the PoR challenge c_i before d_{i-1}, which violates the unpredictability of the random oracle $\mathcal{H}$; or generate the PoR response v_i before c_i, which violates the unpredictability of PoR; or generate the VDF input u_i before v_i, which violates the unpredictability of the random oracle $\mathcal{G}$; or generate the VDF output d_i before u_i, which violates the sequentiality of VDF.

For Claim 2), we prove that the length of each time slot $[T_i, T_{i+1})$ is longer than t'. By the unpredictability of the random oracle, the output of the VDF d_i must be generated before the time point T_{i+1}. On the other hand, the PoR response v_i must be generated via the PoR on the challenge c_i after the time point T_i. Therefore, a VDF function must be evaluated within the time slot $[T_i, T_{i+1})$. By the sequentiality of VDF, the length of $[T_i, T_{i+1})$ must be longer than t'.

For Claim 3), we prove that the length of each time slot $[T_i, T_{i+1})$ is shorter than $t' + \delta T$. Let us denote the execution time of $\mathsf{PoSt}.\mathcal{P}'$ as T'. By the correctness of the verification algorithm, $T' < (1+\delta)T$. Since we have proved that the length of each time slot $[T_i, T_{i+1})$ is longer than t', the longest slot should be shorter than $(1+\delta)T - (k-1)t' = \delta T + t'$.

For Claim 4), the thing left is to show that the PoR response v_i must have been queried to the random oracle $\mathcal{G}$ in this time slot $[T_i, T_{i+1})$ and the time slot $[T_i, R_i)$ is shorter than δT. On the one hand, the output of the VDF d_i is queried at the time point T_{i+1}. So the input of the VDF u_i must be generated by $\mathsf{PoSt}.\mathcal{P}'$ before the time T_{i+1} according the sequentiality of VDF. By the unpredictability of the random oracle, $\mathcal{G}$ must be queried on input v_i before the time T_{i+1}. On the other hand, according to the unpredictability of PoR mentioned in 2.2, $\mathrm{PoSt}.\mathcal{P}'$ can not figure out the PoR proof v_i before the time point T_i when the PoR challenge c_i is generated. Given all this, v_i must have been queried to the random oracle $\mathcal{G}$ in time slot $[T_i, T_{i+1})$. Furthermore, since the maximum length of $[T_i, T_{i+1})$ and the evaluation time of VDF is longer then t', the time slot $[T_i, R_i)$ is shorter than δT.

For the second step of the proof, we show that given the bunch of configurations for $\mathsf{PoSt}.\mathcal{P}'$ for time slot $[T_i, R_i)$ (or $[T_{i-1}, R_{i-1})$) and the code of the transition function, c_i and v_i can be easily accessed by the $\mathsf{PoSt.Extr}$. Indeed, since both random oracles $\mathcal{H}$ and $\mathcal{G}$ are maintained by the extractor, a cheating PoR prover $\mathsf{PoR}.\mathcal{P}'$ can be constructed by manipulating the output of the random oracle $\mathcal{H}$ as the PoR challenge, rewinding the part of the $\mathsf{PoSt}.\mathcal{P}'$ corresponding to time segment $[T_i, R_i)$ (or $[T_{i-1}, R_{i-1})$) and collecting the queries of the random oracle $\mathcal{G}$ as the PoR response. Since there is a PoR extractor to recover the storage data from $\mathsf{PoR}.\mathcal{P}'$, the soundness proof of PoSt completes. □

5.2 Compact PoSt scheme

Although the above basic PoSt already achieves the purpose of verifying the continuous availability of data, the large proof size makes it impractical for many applications, including the decentralized storage market advocated by Filecoin. One approach could be to let the prover compress all transcripts and prove the validity of each challenge and the final compressed proof using zk-SNARK as in [40]. Unfortunately, the generic method that employs zk-SNARK to prove the corresponding statement incurs a prohibitive cost (both computational and memory-wise); thus, it is practically infeasible.[9]

Next, we describe our compact PoSt. The structure of our construction can be seen in Fig. 1. As in our basic construction, the prover here also executes the sequential PoR schemes where each

[9] The proof time of SNARK is proportional to the circuit size, and the memory cost grows even faster. According to the latest report [11] by Ben-Sassone et al., the time cost for generating the proof is roughly 0.1 ms multiplied by the number of circuit gates. If one wants to store the data for one month and check it every hour, the basic PoSt prover is required to compute 720 PoRs. Even if we adopt the simplest PoR, e.g., the HMAC based one [31], the storage provider still needs to compute more than 2^{27} hashes for 64MB data, and the gates for one hash function are at least 2^{13} according to the estimation by Bernstein [13]. Therefore, the total proof generation time, if we use SNARKs, would require more than five years. If one turns the entire prove procedure into a giant circuit, the circuit size becomes larger than 2^{37}, and the memory cost would be about 1TB already for all HMAC-based PoRs.

next challenge is the output of the delay function, and the verifier gives only the first challenge. But instead of verifying each PoR instance, we let the verifier perform the same work as the prover during the storing phase, except that he can compute the delay function much faster thanks to the trapdoor. Therefore, the verifier only needs to check that all PoR challenges and responses from the prover are the same as the ones he computed. Hence, the final proof consists of the hash of all PoR challenges and responses, and the tag denotes the hash image of this value. Besides, challenges in our compact PoSt can only be generated by the data owner; therefore, it provides only public validity, but not public variability according to the formulation in Section 4.

Comparing with the previous construction, the communication size of our scheme is constant. But the scheme is stateful. That means that the number of challenges is bounded and the verifier must record the previous challenge history. If the bound is reached, the data owner must retrieve the data and rerun the storing procedure.

Our compact PoSt construction uses a publicly verifiable stateful or stateless PoR scheme and a TDF construction. The PoR scheme consists of a tuple of algorithms

$$(\mathsf{PoR.Kg}, \mathsf{PoR.Store}, \mathsf{PoR}.\mathcal{P}, \mathsf{PoR}.\mathcal{V}),$$

where $\mathsf{PoR}.\mathcal{V} = (\mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}, \mathsf{PoR}.\mathcal{V}_{\mathsf{valid}})$. Specifically, the PoR scheme should be deterministic, which means there is only one valid response to a specific challenge, and unpredictable, which means the response can not be predicted in advance before viewing the challenge. The TDF scheme with δ-evaluation time consists of a tuple of algorithms $(\mathsf{TDF.Setup}, \mathsf{TDF.Eval}, \mathsf{TDF.TrapEval})$. Moreover, $\mathcal{H}$, $\mathcal{H}_1$, $\mathcal{H}_2$, $\mathcal{H}_3$ and $\mathcal{G}$ are hash functions which can be viewed as random oracles. D is the data to be stored. $\mathsf{SE} = (\mathsf{SKg}, \mathsf{Enc}, \mathsf{Dec})$ is a semantically secure symmetric-key encryption. Let l denote the bound of interaction times between the prover and verifier. The compact PoSt scheme needs a global timer which is initiated by the data owner but can be seen by everyone.

The compact PoSt is parameterized by the storage time T and the checking frequency t'. As before, the data is checked more frequently than the requested frequency. In the construction, the delay parameter of TDF is set as t' where $t > t' + 2\delta T$. Here T, t and t' are all measured by the unit steps of a Turing machine, which corresponds to the CPU clock. See Section 6.2 for more detailed discussions.

The compact PoSt scheme is as follows.

- $\mathsf{cPoSt.Kg}(\lambda, t, T)$: Invoke $\mathsf{PoR.Kg}(\lambda)$ to generate a PoR key pair $\mathsf{PoR}.pk$ and $\mathsf{PoR}.sk$. Also generate a secret key $\mathsf{SE}.sk$ for symmetric encryption via $\mathsf{SKg}(\lambda)$. Choose t' as the largest number of ITM steps such that $t' < t - 2\delta T$ and $k = T/t'$ is an integer. Run $\mathsf{TDF.Setup}(\lambda, s_0)$ to generate the TDF's public parameter $\mathsf{TDF}.pp$ and trapdoor $\mathsf{TDF}.tr$ where the time cost on the server processor is t'. The public key $pk = (\mathsf{PoR}.pk, \mathsf{TDF}.pp, T, k)$, while the secret key $sk = (\mathsf{PoR}.sk, \mathsf{SE}.sk, \mathsf{TDF}.tr)$.
- $\mathsf{cPoSt.Store}(pk, sk, l, D)$: Take as input the public key pk, the secret key sk, the expected storage time T, the bounded number l and a file $D \in \{0,1\}^*$, then run Algorithm 3 to generate the encoded file D^* and the tag tg. Intuitively, the data owner sequentially computes PoR instances where the next PoR challenge is generated from the previous PoR proof via hash functions and TDF trapdoor evaluations. Then he keeps the hash values of all the PoR challenges and responses together for further verification.
- $\mathsf{cPoSt}.\mathcal{V} = (\mathsf{cPoSt}.\mathcal{V}_{\mathsf{cha}}, \mathsf{cPoSt}.\mathcal{V}_{\mathsf{valid}})$:
  - $\mathsf{cPoSt}.\mathcal{V}_{\mathsf{cha}}(pk, sk, tg, state)$: Keep a variable $state$ to record the number of interactions. If $state = i < l$, $\mathsf{cPoSt}.\mathcal{V}_{\mathsf{cha}}$ uses $\mathsf{SE}.sk$ to decrypt the ciphertext C in the tag tg, gets the corresponding challenge c_i and sends it to the prover. Meanwhile, reset the timer as 0 and increment $state$.
  - $\mathsf{cPoSt}.\mathcal{V}_{\mathsf{valid}}(pk, tg, state, p)$: When receiving the PoSt proof p from the prover, $\mathsf{cPoSt}.\mathcal{V}_{\mathsf{valid}}$ first check the current timer. If the timer is shorter than T or longer than $(1+\delta)T$, $\mathsf{cPoSt}.\mathcal{V}_{\mathsf{valid}}$ outputs $reject$, otherwise $\mathsf{cPoSt}.\mathcal{V}_{\mathsf{valid}}$ checks whether $\mathcal{H}_3(p) = tg_i$ for $state = i$. If it is true, $\mathsf{cPoSt}.\mathcal{V}_{\mathsf{valid}}$ outputs $accept$, otherwise outputs $reject$.
- $\mathsf{cPoSt}.\mathcal{P}(c_0, pk, tg, D^*)$: After receiving a challenge c_0 from the verifier, the prover runs Algorithm 4 to generate a proof p. Intuitively, the prover sequentially computes PoR instances where the next PoR challenge is generated from the previous PoR proof via hash functions and TDF evaluations without trapdoor. Then he hashes all the PoR challenges and responses to generate the final PoSt proof.

Algorithm 3 The compact PoSt storing algorithm $\mathsf{cPoSt.Store}$

Require: The public key pk, the secret key sk, the number l and a file $D \in \{0,1\}^*$
Ensure: D^* for storage and a tag tg

Parse $pk = (\mathsf{PoR}.pk, \mathsf{TDF}.pp, T, k)$
Parse $sk = (\mathsf{PoR}.sk, \mathsf{SE}.sk, \mathsf{TDF}.tr)$
$(D^*, tg^*) \leftarrow \mathsf{PoR.Store}(\mathsf{PoR}.pk, \mathsf{PoR}.sk, D)$// *Run the PoR storing*
for $j = 1$ to l **do**
    $c_{0,j} \leftarrow \mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}(\mathsf{PoR}.pk, tg^*)$
    for $i = 0$ to $k-1$ **do**
        $v_{i,j} \leftarrow \mathsf{PoR}.\mathcal{P}(\mathsf{PoR}.pk, c_{i,j}, D^*, tg^*)$// *Generate a PoR proof.*
        $u_{i,j} = \mathcal{G}(v_{i,j})$
        $d_{i,j} \leftarrow \mathsf{TDF.TrapEval}(\mathsf{TDF}.pp, \mathsf{TDF}.tr, u_{i,j})$// *Use the trapdoor to evaluate TDF efficiently*
        $c_{i+1,j} = \mathcal{H}(d_{i,j})$
    $v_{k,j} \leftarrow \mathsf{PoR}.\mathcal{P}(\mathsf{PoR}.pk, c_{k,j}, D^*, tg^*)$
    $c_j = \mathcal{H}_1(c_{0,j}, \ldots, c_{k,j})$
    $v_j = \mathcal{H}_2(v_{0,j}, \ldots, v_{k,j})$
    $tg_j = \mathcal{H}_3(c_j, v_j)$
$C = \mathsf{Enc}_{\mathsf{SE}.sk}(c_1, \ldots, c_l)$
$tg = (C, tg^*, tg_1, \ldots, tg_l)$
return D^* and tg

Algorithm 4 The compact PoSt prove algorithm $\mathsf{cPoSt}.\mathcal{P}$

Require: The initial challenge c_0, the stored data D^* and the tag tg
Ensure: The PoSt proof p

Parse $pk = (\mathsf{PoR}.pk, \mathsf{TDF}.pp, k, T)$
Parse $tg = (C, tg^*, tg_1, \ldots, tg_l)$
for $i = 0$ to $k-1$ **do**
    $v_i \leftarrow \mathsf{PoR}.\mathcal{P}(\mathsf{PoR}.pk, c_i, D^*, tg^*)$// *Generate a PoR proof.*
    $u_i = \mathcal{G}(v_i)$
    $d_i \leftarrow \mathsf{TDF.Eval}(\mathsf{TDF}.pp, u_i)$// *Evaluate TDF without trapdoor*
    $c_{i+1} = \mathcal{H}(d_i)$
$v_k \leftarrow \mathsf{PoR}.\mathcal{P}(\mathsf{PoR}.pk, c_k, D^*, tg)$
$c = \mathcal{H}_1(c_0, \ldots, c_k)$
$v = \mathcal{H}_2(v_0, \ldots, v_k)$
return $p = (c, v)$

Correctness Note that the final PoSt proof procedure includes k VDF evaluations where $k = T/t'$. Since we assume that the time of evaluating one VDF with δ-evaluation time is shorter than $(1+\delta)t'$, the total time cost for an honest prover to generate a PoSt proof is less than $(1+\delta)T$. Hence the correctness of our compact PoSt directly follows from that of the PoR and VDF schemes.

Soundness The proof strategy of the Compact PoSt scheme is similar to the proof in Theorem 1. In general, the verification algorithm of the compact PoSt requires the prover to compute all PoR challenges and responses and evaluate the TDFs as in the storing phase, so we can easily conclude that all the PoR responses are valid and the TDFs are evaluated as expected. Because of the unpredictability of PoR and the sequentially of TDF, the PoR proofs must be generated sequentially. Moreover, the random time epoch with length longer than t must contain at least one PoR execution, and both the input and output of the PoR can be located via random oracles $\mathcal{G}$ and $\mathcal{H}$, respectively. Therefore, we can invoke the PoR extractor to recover the data.

Theorem 2. *PoR is a PoR scheme with ϵ-soundness and unpredictability. TDF is a TDF scheme with δ-evaluation time. Assume that the time cost for TDF is at least t' and $t > t' + 2\delta T$, then the above compact PoSt scheme is stateful with l permitted challenges and has ϵ-soundness.*

Proof. Assume that an adversary $\mathcal{A}$ outputs a cheating prover $\mathrm{cPoSt}.\mathcal{P}'$ in the *setup* game. To show the soundness of the scheme, we need to construct the extractor $\mathsf{cPoSt.Extr} = (\mathsf{cPoSt.Extr}_1, \mathsf{cPoSt.Extr}_2)$ to recover the data D from $\mathsf{PoSt}.\mathcal{P}'$. In general, the verification algorithm of the compact PoSt

requires the prover to compute all PoR challenges and responses and evaluate the TDFs as in the setup phase, so we can easily conclude that all the PoR responses are valid and the TDFs are evaluated as deemed. Therefore, we can use the strategy similar to Theorem 1 to construct $\mathsf{cPoSt.Extr}$.

Specifically, let T_0 and T_k be the starting and ending time points for running $\mathsf{PoSt}.\mathcal{P}'$. Similar to Theorem 1, we need to set $T_1, \ldots, T_{k-1}$ and $R_0, \ldots, R_{k-1}$ as the points of respective queries to the random oracle $\mathcal{H}$ and $\mathcal{G}$ for $\mathsf{PoSt}.\mathcal{P}'$, so that:

Claim 1) T_i must precede T_{i+1},
Claim 2) the length of each time slot $[T_i, T_{i+1})$ is longer than t',
Claim 3) the length of each time slot $[T_i, T_{i+1})$ is shorter than $t' + \delta T$,
Claim 4) each R_i belongs to the time slot $[T_i, T_{i+1})$ and the time slot $[T_i, R_i)$ is shorter than δT.

Therefore, the random time epoch with length $t > t' + 2\delta T$ chosen by $\mathsf{PoSt.Extr}_1$ must contain at least one interval $[T_i, R_i)$ for some i. Hence a cheating PoR prover $\mathsf{PoR}.\mathcal{P}'$ can be constructed by $\mathsf{PoSt.Extr}_2$ since both random oracles $\mathcal{H}$ and $\mathcal{G}$ are maintained and hence able to be manipulated by the extractor. □

6 Instantiations

Although we have provided a generic framework for PoSt with asymptotically compact proofs, we still need to discuss the practical instantiations of our two main building blocks: PoR and VDF(TDF).

6.1 An efficient PoR instantiation

When taking into account the concrete efficiency, we note that the verification phase of our compact PoSt is extremely efficient since only one evaluation of a hash function is involved. The cost of its proof phase is inherent since the server inevitably needs to keep computing the delay function. However, the cost of the setup phase affects the overall efficiency. Based on our design, the data owner needs to compute all PoRs and TDFs sequentially. Since he holds the trapdoor of the TDF, computing PoR proofs becomes the main burden when the data size is large. For instance, if the expected storage time is one month and the audit frequency is every hour, the setup phase consists of about 720 PoRs. In this case, the classic PoR based on bilinear pairings [44] or RSA group [8] are not satisfactory.

Stateful PDP/PoRs achieve higher efficiency since they can be built from symmetric-key primitives [31]. However, stateless PoR schemes only support a very limited (usually constant) number of challenges. At first glance, these schemes are not suitable for the compact PoSt; however, we observe that the limited number of challenges is due to the verification algorithm of PoR, which by our design, we never invoke in our compact PoSt construction. The prove algorithm of stateless PoRs does support a polynomial number of challenges. Accordingly, we can adopt a simple stateful PoR, as in [31], in our PoSt construction.

Let $\mathcal{H}$ be a HMAC and $\mathcal{G}$ be a hash function. The PoR scheme is as follows:

- $\mathsf{PoR.Kg}(\lambda)$: Taking as input the security parameter λ, randomly choose the secret key sk as a sequence of bit strings from $\{0,1\}^\lambda$, i.e., $sk = (r_1, \ldots, r_n) \in \{0,1\}^{\lambda \times n}$. Note that no public key is needed in this scheme.
- $\mathsf{PoR.St}(sk, D)$: Take as input a secret key $sk = (r_1, \ldots, r_n)$ and a file $D \in \{0,1\}^*$, then compute the MAC $p_i = \mathcal{H}(r_i, D)$ w.r.t. the key r_i for $i = 1, \ldots, n$. Let $t_i = \mathcal{G}(p_i)$ and the PoR tag $tg = (t_1, \ldots, t_l)$.
- $\mathsf{PoR}.\mathcal{V} = (\mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}, \mathsf{PoR}.\mathcal{V}_{\mathsf{verify}})$:
  - $\mathsf{PoR}.\mathcal{V}_{\mathsf{cha}}(sk, state)$: For the $state = i < n$, send the random string $c_i = r_i$ to the verifier.
  - $\mathsf{PoR}.\mathcal{V}_{\mathsf{verify}}(c_i, tg)$: Given the response p_i from the prover when the state is i, if $t_i = \mathcal{G}(p_i)$, output *accept*, otherwise *reject*.
- $\mathsf{PoR}.\mathcal{P}(c_i, D)$: Given the challenge c_i, compute the MAC value $p_i = \mathcal{H}(c_i, D)$ w.r.t. the key c_i.

In practice, $\mathcal{H}$ can be instantiated via the HMAC with SHA-3 [26]. The above scheme is a secure PoR when we model $\mathcal{H}$ and $\mathcal{G}$ as random oracles since the extractor can easily recover data from the random oracle queries. Similarly, we achieve the unpredictability of the PoR from the properties of the random oracle.

6.2 Instantiations of the delay function

In the described PoSt constructions, the time is measured by the number of the ITM steps, which aims to mimic the CPU clock. But for a practical system, we must set the concrete parameters for the VDF/TDFs according to the time and verification frequency. Indeed, as in computational timestamping [14] or other applications of VDFs, a reasonable estimation of the attacker's evaluation speed of the delay function is needed for PoSt. Three items should be considered: 1) choosing the proper instantiation of the delay functions, 2) setting the concrete parameters for VDF/TDF, and 3) making the estimation of the forced delay time as accurate as possible.

First of all, choosing a proper instantiation of the VDF/TDF for our PoSt needs special care. Given the time T and checking frequency t as chosen by the data owner, our PoSt constructions require the parameter δ of the delay function to be smaller than $\frac{t}{2T}$; otherwise, the key generation algorithm cannot find a suitable t'. Therefore, to achieve a small enough δ, it is recommended to instantiate the VDF/TDF schemes as the tight ones in [25, 48], which can be evaluated in sequential steps $t + O(1)$ with an honest prover using $O(\log(t))$ processors and space.

Note that in our PoSt model, the storage time T, and the frequency parameter t are measured by the number of steps of the ITM. Since existing delay functions are evaluated via specific unit operations, such as modular squaring [14, 48, 38], a more practical method would be to choose the number of unit operations directly. Specifically, given the parameter δ for the delay function, the client can choose the desired storage time $\mathbf{T}$ and checking frequency $\mathbf{t}$ (both measured in minutes), then choose a time $\mathbf{t}'$ such that $\mathbf{t}' < \mathbf{t} - 2\delta\mathbf{T}$ and $k = \mathbf{T}/\mathbf{t}'$ is an integer. After that, the client estimates the time of each unit operation and find a number s_0 such that the time spent to compute s_0 unit operations is the closest to (but smaller than) $(1 + \delta)\mathbf{t}'$. Consequently, the honest prover can sequentially run s_0 operations for one delay function and compute a valid PoSt within time $(1 + \delta)\mathbf{T}$. On the other hand, any malicious server would spend at least $\mathbf{t}'$ time for one delay function, so that the largest interval between two PoRs must be less than $\mathbf{t}$.

Note that the adversary may still deploy some special hardware to speed up the computation of unit operations, which would violate the security guarantee. Indeed, such a concern was recognized by the community, and Ethereum/Filecoin invested significant resources in developing specialized hardware and in optimizing implementation runtimes [32, 41] to obtain the fastest implementation of the delay function. In our setting, it is more rational to focus on providing an excellent storage service than investing in an arms race with organizations that are centered on hardware manufacturing. Nevertheless, more strategies are needed when estimating concrete parameters considering this aspect.

7 Evaluation

Implementation. To evaluate the performance of our scheme, we implemented a prototype in C++, employing the Crypto++ library Version 8.2 [22] for cryptographic operations. The experiments were run on a MacBook Pro with 32 GB 2400MHz DDR4 memory and a 2.9 GHz Intel Core i9 CPU. Specifically, we estimate the cost for different storage times (1 to 5 months) and various file sizes (32MB to 256MB) and require the prover to check up the file according to different frequencies. All hash functions in our algorithms are instantiated with SHA-3 [26]. As for the files, we use randomly generated files of different sizes. The numbers are the averages of 5 runs.

Setup cost. The main cost of our compact PoSt is the setup algorithm, which consists of two parts. The first part is to compute the PoR in Section 2.2. The second part is to evaluate the TDF with the trapdoor. We observe that the time cost for the trapdoor evaluation is almost the same for the RSA based TDF construction [48], no matter how long the delay time is. Therefore, we estimate the cost of $\mathsf{TDF.TrapEval}$ by computing the modulus exponentiation in an RSA group for a random exponent. The RSA modulus used here is of size 1024 bits. The results of our experiments with four different data available time $\mathbf{T}$ (delay function parameter δ and checking frequency $\mathbf{t}$) and varying file sizes are depicted in Fig. 3 (Fig. 4 and Fig. 5 respectively). According to the experimental results, the setup algorithm for a file of size 64 MB, which is supposed to be stored for 1 month, takes about 4 minutes.

One may observe that in Fig.3, the setup time cost increases a little bit faster than linear to the storage time. This is because t' gets smaller for larger T due to $t' < t - 2\delta T$, so the honest prover

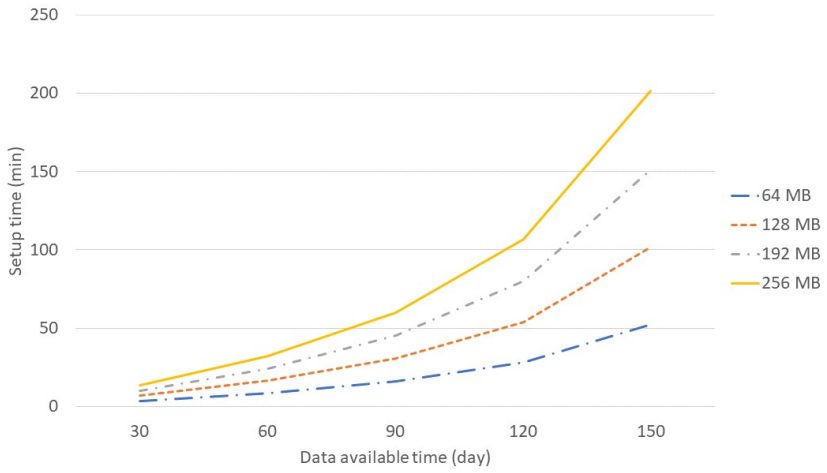


Fig. 3. The compact PoSt setup times for various file sizes and different data available periods, with hourly check up policy and $\delta = 0.0001$ of the delay function.

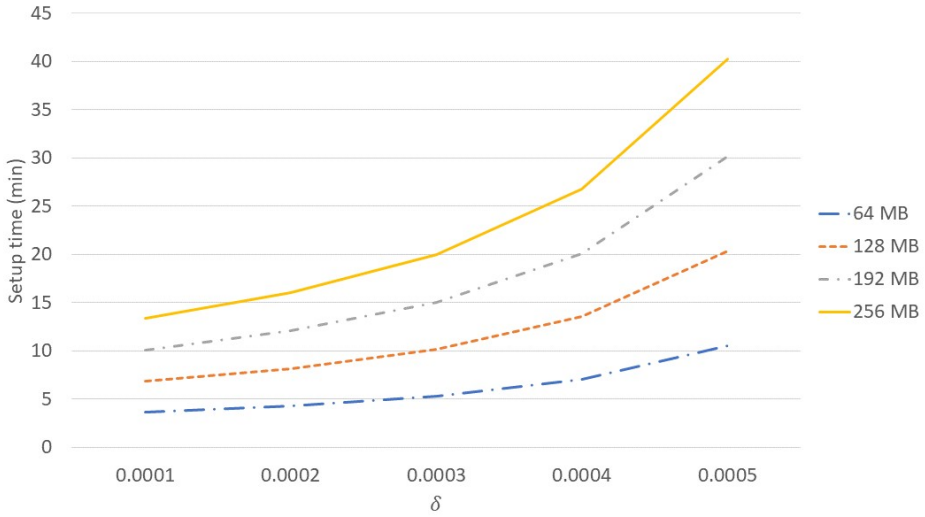


Fig. 4. The compact PoSt setup times for various file sizes and different δ of the delay function, with hourly check up policy and 30 days data available period.

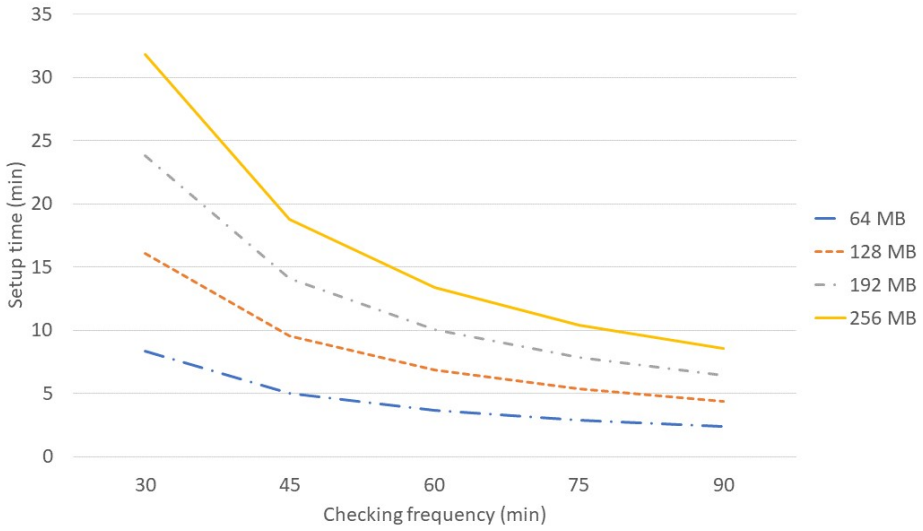


Fig. 5. The compact PoSt setup times for various file sizes and different checking frequency, with $\delta = 0.0001$ and 30 days data available period.

has to check the data more frequently to guarantee the same level of proved checking frequency for the adversary even for same t. This is not hard to imagine since when the storage time is longer, it is harder to achieve the same accuracy for the delay time.

Verification cost. It is easy to see that our verification only computes a hash; thus, the cost can be considered negligible. If a smart contract is instantiated by our compact PoSt, to check whether the submitted string p is the hash pre-image of the fixed value tg_i only costs 36 gas (worth less than 0.0001 USD[1]) in Ethereum if the hash is instantiated via SHA-3.

Proof cost. The cost of computing delay functions is inherent for the PoSt. Besides, the prover is doing PoR proofs, whose cost can be ignored compared to the delay time.

Discussions. The setup time of our PoSt scheme may be noticeable, but there could be multiple ways to optimize the performance further. (1) Setup only needs to be done once for every storage period. Thus pre-computation can always be performed except for the first storage period. For example, consider the case where the user initially stores his files for half a year, and he would like to extend the contract for another half a year. The setup can be finished before the second storage phase begins. (2) One major factor affecting the setup times is hashing large files (used in the simple PoR scheme). An optimized approach, such as using parallel processors, would improve setup times considerably.

8 Conclusions

In this paper, we systematically studied the notion of *Proof of Storage-time*, which enables a client to efficiently verify that outsourced data is continuously retrievable from a storage provider. We proposed formal definitions and presented efficient constructions with rigorous security analyses. Our result is the first step towards studying advanced concepts of continuous data availability. Several open problems remain, including making PoSt stateless and without relying on any trapdoor, reducing setup cost, supporting proof of replication, and dynamic updates.

Acknowledgment

The authors were supported in part by a grant from Protocol Labs/Filecoin. We thank Jeremiah Blocki and the anonymous reviewers for valuable comments. Long Chen and Qiang Tang are also supported in part by JDDigits via the JDD-NJIT-ISCAS Joint Blockchain Lab.

References

1. ETH Gas Station. https://ethgasstation.info/. Accessed: January 12, 2020.
2. Tencent Cloud user claims $1.6 million compensation for data loss. https://technode.com/2018/08/06/tencent-cloud-user-claims -1-6-million-compensation-for-data-loss/ . Accessed: 2019-01-31.
3. Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 99–130. Springer, 2018.
4. Frederik Armknecht, Ludovic Barman, Jens-Matthias Bohli, and Ghassan O Karame. Mirror: Enabling proofs of data replication and retrievability in the cloud. In *USENIX Security Symposium*, pages 1051–1068, 2016.
5. Frederik Armknecht, Jens-Matthias Bohli, Ghassan O Karame, Zongren Liu, and Christian A Reuter. Outsourced proofs of retrievability. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 831–843. ACM, 2014.
6. Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In *International Conference on Security and Cryptography for Networks*, pages 538–557. Springer, 2014.
7. Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Osama Khan, Lea Kissner, Zachary Peterson, and Dawn Song. Remote data checking using provable data possession. *ACM Trans. Inf. Syst. Secur.*, 14(1), June 2011.
8. Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *CCS'07*. ACM, 2007.
9. Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *SecureComm*, page 9. ACM, 2008.
10. Thomas Beaton. Top 10 healthcare mobile apps among hospital, health systems. https://mhealthintelligence.com/news/top-10-healthcare-mobile-apps-among-hospital-health-systems , 2017.
11. Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 781–796, 2014.
12. David Bermingham and Joey D'Antoni. Controlling costs in the cloud for high-availability applications. http://www.dbta.com/Editorial/Trends-and-Applications/Controlling-Costs-in-the-Cloud-for-High-Availability-Applications-127914.aspx, 2019.
13. Daniel J Bernstein. Sha-3 interoperability.
14. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual International Cryptology Conference*, pages 757–788. Springer, 2018.
15. Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. 2018.
16. Adam Boudjemaa. Decentralized cloud storage is changing the face of the internet. https://hackernoon.com/decentralized-cloud-storage-how-it-will-change-the-face-of-the-internet-22-np1f2349h, 2019.
17. Kevin D Bowers, Ari Juels, and Alina Oprea. Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198. ACM, 2009.
18. Kevin D Bowers, Ari Juels, and Alina Oprea. Proofs of retrievability: Theory and implementation. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 43–54. ACM, 2009.
19. Randal Chilton Burns and Inderpal Singh Narang. Continuous availability and efficient backup for externally referenced objects, July 11 2000. US Patent 6,088,694.
20. David Cash, Alptekin Küpçü, and Daniel Wichs. Dynamic proofs of retrievability via oblivious ram. *Journal of Cryptology*, pages 1–26, 2015.
21. R. Curtmola, O. Khan, R. Burns, and G. Ateniese. Mr-pdp: Multiple-replica provable data possession. In *ICDCS'08*, pages 411–420. IEEE, 2008.
22. Dai, Wei. Crypto++ Library 8.2. https://www.cryptopp.com, 2019.
23. Ivan Damgård, Chaya Ganesh, and Claudio Orlandi. Proofs of replicated storage without timing assumptions. In *Annual International Cryptology Conference*, pages 355–380. Springer, 2019.
24. Yevgeniy Dodis, Salil Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *Theory of Cryptography Conference*, pages 109–127. Springer, 2009.
25. Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini. Tight verifiable delay functions.
26. Morris J Dworkin. Sha-3 standard: Permutation-based hash and extendable-output functions. Technical report, 2015.
27. Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In *Annual Cryptology Conference*, pages 585–605. Springer, 2015.

28. Ben Fisch. Tight proofs of space and replication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 324–348. Springer, 2019.
29. Martin Fuerderer and Ajay Gupta. High availability data replication set up using external backup and restore, March 31 2005. US Patent App. 10/850,781.
30. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.
31. Ari Juels and Burton S Kaliski Jr. PORs: Proofs of retrievability for large files. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 584–597. ACM, 2007.
32. Christine Kim. Ethereum foundation and others weigh $ 15 million bid to build 'randomness' tech. https://www.coindesk.com/ethereum-foundation-weighs-15-million-bid-to-build-randomness-tech, 2019.
33. Tal Moran and Ilan Orlov. Simple proofs of space-time and rational proofs of storage. In *Annual International Cryptology Conference*, pages 381–409. Springer, 2019.
34. Harriet Morrill, Mark Beard, and David Clitherow. Achieving continuous availability of ibm systems infrastructures. *IBM Systems Journal*, 47(4):493–503, 2008.
35. NetApp. Cloud Volumes ONTAP - enterprise data management solution. https://cloud.netapp.com/ontap-cloud. Accessed: 2019-09-14.
36. Maura B Paterson, Douglas R Stinson, and Jalaj Upadhyay. Multi-prover proof of retrievability. *Journal of Mathematical Cryptology*, 12(4):203–220, 2018.
37. Krzysztof Pietrzak. Proofs of catalytic space. In *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
38. Krzysztof Pietrzak. Simple verifiable delay functions. *IACR Cryptology ePrint Archive*, 2018:627, 2018.
39. Bertrand Portier. Always on: Business considerations for continuous availability. http://www.redbooks.ibm.com/redpapers/pdfs/redp5090.pdf, 2014. ©Copyright IBM Corp. 2014.
40. Protocol Labs. Filecoin: A decentralized storage network. https://filecoin.io/filecoin.pdf, 2018. Accessed: 2019-01-31.
41. Protocol Labs. Collaboration with the ethereum foundation on vdfs. https://filecoin.io/blog/collaboration-on-vdfs/, 2019.
42. Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.
43. Daniel Rubino. Continuous availability. https://www.dellemc.com/en-us/glossary/continuous-availability.htm. Accessed: 2019-08-25.
44. Hovav Shacham and Brent Waters. Compact proofs of retrievability. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 90–107. Springer, 2008.
45. Elaine Shi, Emil Stefanov, and Charalampos Papamanthou. Practical dynamic proofs of retrievability. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 325–336. ACM, 2013.
46. Jonathan Shieber. Google cloud is down, affecting numerous applications and services. https://techcrunch.com/2019/06/02/google-cloud-is-down-affecting-numerous-applications-and-services/ , 2019.
47. Qian Wang, Cong Wang, Jin Li, Kui Ren, and Wenjing Lou. Enabling public verifiability and data dynamics for storage security in cloud computing. In *European symposium on research in computer security*, pages 355–370. Springer, 2009.
48. Benjamin Wesolowski. Efficient verifiable delay functions. *IACR Cryptology ePrint Archive*, 2018:623, 2018.
49. Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.
50. Qingji Zheng and Shouhuai Xu. Fair and dynamic proofs of retrievability. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 237–248. ACM, 2011.
51. Qingji Zheng and Shouhuai Xu. Secure and efficient proof of storage with deduplication. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 1–12. ACM, 2012.

Proof of Replication
Technical Report (WIP)

Juan Benet[1] David Dalrymple Nicola Greco[1]

[1]Protocol Labs

July 27, 2017

Abstract

We introduce *Proof-of-Replication* (PoRep), a new kind of *Proof-of-Storage*, that can be used to prove that some data $\mathcal{D}$ has been *replicated* to its own uniquely dedicated physical storage. Enforcing unique physical copies enables a verifier to check that a prover is not deduplicating multiple copies of $\mathcal{D}$ into the same storage space. This construction is particularly useful in *Cloud Computing* and *Decentralized Storage Networks*, which must be transparently verifiable, resistant to Sybil attacks, and unfriendly to outsourcing.

This work (a) reviews *Proofs-of-Storage* and motivates use cases; (b) defines the novel *Proofs-of-Replication*, which can be *publicly verifiable*, *transparent*, *authenticated*, and *time-bounded*; (c) shows how to chain *Proofs-of-Replication* to establish *useful Proofs-of-Spacetime*.

Work in Progress. This is a work in progress Technical Report from Protocol Labs. Active research is under way, and new versions of this paper will appear. For comments and suggestions, contact us at research@filecoin.io

1 Motivation and Background on *Proofs-of-Storage*

This section provides backgrounds and classifications of different *Proofs-of-Storage* and related proofs, and motivates the need for *Proofs-of-Replication*. Throughout this section, we explain the distinction between different proofs using a prover, $\mathcal{P}$, that is attempting to convince a verifier, $\mathcal{V}$, that $\mathcal{P}$ is storing some data, $\mathcal{D}$. $\mathcal{V}$ issues a challenge, c, to $\mathcal{P}$ who answers it with a corresponding proof π^c, according to the scheme in question. Proof schemes vary in their properties, their utility, and in whether $\mathcal{D}$ is useful outside the protocol or is a random string with no external utility.

1.1 Common Properties

We use the following properties, common to various proving schemes:

- (*Privately Verifiable*) A scheme is *privately verifiable* if $\mathcal{V}$ is a user with a secret verifying key generated during setup, or any other party that shares such secret key with the user. These schemes are useful in *Cloud Computing* settings, where users wish to outsource storage of data to servers and perhaps outsource verifying to a trusted verifier. As of this work, most *Proof-of-Storage* (PoS) schemes are *privately verifiable*.
- (*Publicly Verifiable*) A scheme is *publicly verifiable* if $\mathcal{V}$ can be any party with access to public data (e.g. a verifying key), but no access to the original data, or secret information generated during scheme setup. *Publicly verifiable* schemes are very useful in *Decentralized Storage Network* settings, where a verifier may be new participants who have access only to public data as context of previous proof scheme setups.

- (*Transparent*) A scheme is *transparent* if there is no extra information, sk, that enables any $\mathcal{P}$ to generate a valid proof without having data, $\mathcal{D}$. This means that there is no sk with which a malicious prover can generate proof $\pi^* = \mathsf{ForgeProof}(c, sk)$ such that $1 = \mathsf{Verify}(c, \pi^*)$ for a $\mathcal{P}$-chosen c. *Transparent* schemes are necessary in *Decentralized Storage Networks*, where provers may also be verifiers or users. While many PoS schemes require trusting a user or verifier to generate secret keys, *Transparent* PoS schemes do not.

- (*Retrievability*) A scheme supports *retrievability* if it is possible for $\mathcal{V}$ to extract and reconstruct $\mathcal{D}$ merely by issuing many challenges c to $\mathcal{P}$ and aggregating corresponding proofs π^c. See PoRet.

- (*Dynamic*) A PoS scheme is *dynamic* if it enables the user $\mathcal{V}$ to *dynamically update* data $\mathcal{D}$ to $\mathcal{D}$' stored at server $\mathcal{P}$, to support mutable data without requiring a completely new setup. *Dynamic* PoS schemes are very useful in *Cloud Storage* settings, and systems with large, frequently mutable data without need for version history.

- (*Non-Outsourceable*) A scheme is *non-outsourceable* if $\mathcal{P}$ cannot outsource her work to some other prover $\mathcal{P}^*$ (e.g. storage, work, or proof-generation) and convince $\mathcal{V}$ that $\mathcal{P}$ did the work. *Non-outsourceable* schemes are useful in *Cloud Computing*, *Cryptocurrency*, and *Decentralized Storage Network* settings, where $\mathcal{P}$ may be rewarded for providing space, and the users or $\mathcal{V}$ wish to ensure $\mathcal{P}$ is actually providing the service.

- (*Authenticated*) A scheme is *authenticated* if the the identity of a prover can be verified during a proof verification. For example a digital signature might be required as part of generating π^c to prove identity, pk_i. *Authentication* can help make schemes *non-outsourceable*, since a prover would have to reveal secret identifying information (e.g. their private key) to the outsourced provider.

- (*Time-Bounded*) A proving scheme is *time-bounded* if a proof is only valid during a span of time. For example, some schemes declare that $\mathcal{P}$ must generate a valid proof, π, within a certain time after receiving challenge, c. If $\mathcal{P}$ delays beyond some scheme-specific time bound, then the proof is no longer valid, as $\mathcal{P}$ had enough time to forge it.

- (*Useful*) A scheme is *useful* if it can achieve separate useful work or useful storage as part of its operation or as a side-effect. For example, storing and verifying a verifier-chosen $\mathcal{D}$ (PDP, PoRet, PoRep) is *useful*, whereas storing and verifying a randomly-generated $\mathcal{D}$ (PoSpace) is not *useful*.

1.2 Kinds of Proofs

Here we give an overview of various kinds of proving schemes, in particular *Proofs-of-Storage* and its variants:

- ***Provable Data Possession*** (PDP) schemes [1] allow user $\mathcal{V}$ to send data $\mathcal{D}$ to server $\mathcal{P}$, and later $\mathcal{V}$ can repeatedly check whether $\mathcal{P}$ is still storing $\mathcal{D}$. PDPs are useful in cloud storage and other storage outsourcing settings. PDPs can be either *privately-verifiable* or *publicly-verifiable*, and *static* or *dynamic*. A wide variety of PDP schemes exist.

- ***Proof-of-Retrievability*** (PoRet) schemes [8], [12] are similar to PDPs, but also enable extracting $\mathcal{D}$, namely they offer *retrievability*. PDPs allow the verifier $\mathcal{V}$ to check that $\mathcal{P}$ is still storing $\mathcal{D}$, but $\mathcal{P}$ may submit valid PDP proofs yet hold $\mathcal{D}$ hostage and never release it. PoRs solves this problem by making the proofs themselves leak pieces of $\mathcal{D}$ so that $\mathcal{V}$ can issue some number of challenges and then reconstruct $\mathcal{D}$ from the proofs.

- ***Proof-of-Storage*** (PoS) schemes allow a user $\mathcal{V}$ to outsource the storage of data $\mathcal{D}$ to a server $\mathcal{P}$ and then repeatedly check if $\mathcal{P}$ is still storing $\mathcal{D}$. PDPs and PoRets were independently introduced around the same time in 2007. Since then, the concept of *Proofs-of-Storage* generalizes PDPs and PoRets. This work presents PoReps, a new type of PoS.

- ***Proof-of-Replication*** (PoRep) schemes (this work) are another kind of PoS that additionally ensure that $\mathcal{P}$ is dedicating unique physical storage to storing $\mathcal{D}$. $\mathcal{P}$ cannot pretend to store $\mathcal{D}$ twice and deduplicate the storage. This construction is useful in *Cloud Storage* and *Decentralized Storage Network*

settings, where ensuring a proper level of replication is important, and where rational servers may create Sybil identities and sell their service twice to the same user. PoRep schemes ensure each replica is stored independently. Some PoRep schemes may also be PoRet schemes.

- ***Proof-of-Work*** (PoW) schemes [6] allow prover $\mathcal{P}$ to convince verifier $\mathcal{V}$ that $\mathcal{P}$ has spent some resources. The original use case [6] presented this scheme to allow a server $\mathcal{V}$ to rate-limit usage by asking user $\mathcal{P}$ to do some expensive work per-request. Since then, PoW schemes have been adapted for use in cryptocurrencies, Byzantine Consensus [11], and many other systems. Famously, the Bitcoin network [11] expends a massive amount of energy in a hashing PoW scheme, used to establish consensus and extend the Bitcoin ledger safely.
- ***Proof-of-Space*** (PoSpace) schemes allow prover $\mathcal{P}$ to convince verifier $\mathcal{V}$ that $\mathcal{P}$ has spent some storage resources. PoSpace schemes are PoW schemes where the expended resource is not computation (CPU instructions) but rather storage space. In a sense, a PoS scheme is also a PoSpace, since a PoS implies the use of storage resources.
- ***Proof-of-Spacetime*** (PoSt) schemes [10] allow prover $\mathcal{P}$ to convince verifier $\mathcal{V}$ that $\mathcal{P}$ has spent some "spacetime" (storage space used over time) resources. This is a PoSpace with a sequence of checks over time. A *useful* version of PoSt would be valuable as it could replace other PoW schemes with a storage service. This work introduces such a scheme, based on sequential PoReps.

Definition 1.1. ($\prod^{\mathsf{PoS}}$) This work also defines a simplified PoS proving scheme $\prod^{\mathsf{PoS}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, where:

- $\mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{V} \leftarrow \mathsf{PoS.Setup}(1^\lambda, \mathcal{D})$ where $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{V}$ are scheme-specific setup variables for $\mathcal{P}$ and $\mathcal{V}$ respectively, that depend on the data $\mathcal{D}$, and on a security parameter λ.
- $\pi^c \leftarrow \mathsf{PoS.Prove}(\mathcal{S}_\mathcal{P}, \mathcal{D}, c)$ where c is a challenge, and π^c is a proof that $\mathcal{P}$ has access to $\mathcal{D}$.
- $\{0,1\} \leftarrow \mathsf{PoS.Verify}(\mathcal{S}_\mathcal{V}, c, \pi^c)$ which $\mathcal{V}$ runs to checks whether a proof from $\mathcal{P}$ is correct.

1.3 Motivation for *Proofs-of-Replication*

Consider the following scenarios:

- (*replication*) A user $\mathcal{V}$ wishes to hire server $\mathcal{P}$ to store n independent *copies* of data $\mathcal{D}$; in other words, $\mathcal{V}$ wants a *replication* factor of n. PDP and PoRet schemes do not give $\mathcal{V}$ a way to verify $\mathcal{P}$ is storing these n replicas separately rather than merely pretending to do so.
- (*deduplication*) A user $\mathcal{V}$ asks each of n different servers $\mathcal{P}_0 ... \mathcal{P}_n \in \mathcal{P}$ to store data $\mathcal{D}$. With normal PDP and PoRet schemes, the servers could collude and store $\mathcal{D}$ only once, instead of n times (once each). When issued a challenge, $\mathcal{P}_i$ would only need to retrieve $\mathcal{D}$ from whichever $\mathcal{P}_j$ is actually storing it, calculate the proof, and discard $\mathcal{D}$.
- (*Sybil identities*) A setup very similar to *deduplication* above, but now all servers $\mathcal{P}_0 ... \mathcal{P}_n \in \mathcal{P}$ are secretly just one server, say $\mathcal{P}_0$. The others are Sybil identities.
- (*networks*) A set of users and servers come together to form a *Decentralized Storage Network*, where all participants simulate a unified service that outsources storage to each individual server. Ideally, each individual server could prove they are storing each *replica* of data uniquely, in a *transparent*, and *publicly verifiable* way.

Current PoS schemes do not address these scenarios in full. PDP and PoRet schemes do not prevent a single prover (or group of provers) $\mathcal{P}$ from deduplicating data across multiple user requests. Users with $\mathcal{D}$ can achieve *replication* with PDP and PoRet schemes by deriving a set of encrypted replicas $\mathcal{D}^{sk_i}$, and keeping the mapping and keys secret. However, this is expensive and not *transparent*, which means that in a *Decentralized Storage Network* setting, a user could also play the role of $\mathcal{P}$, have access to some of the replication mappings and keys, and thus deduplicate the storage. We must do better.

2 *Proofs-of-Replication*

We introduce *Proof-of-Replication* (PoRep) schemes, which allow a prover $\mathcal{P}$ to (a) commit to store n distinct *replicas* (physically independent copies) of $\mathcal{D}$, and then (b) convince a verifier $\mathcal{V}$ that $\mathcal{P}$ is indeed storing each of the replicas. We formalize three challenges: *Sybil Attack*, *Outsourcing Attack*, and *Generation Attack*, all concerning *replication*. The ability to surmount these challenges distinguishes PoRep schemes from other PoS schemes. We introduce an adversarial game called RepGame that PoRep schemes must pass to be secure, and the formal definition of PoRep schemes. Finally, we explore a sub-class of PoRep schemes, *Time-Bounded Proofs-of-Replication*, that are significantly easier to realize.

Definition 2.1. (*Sybil Attack*) An attacker $\mathcal{A}$ has Sybil identities $\mathcal{P}_0...\mathcal{P}_n$, and makes each commit to storing a *replica* of $\mathcal{D}$. The attack succceeds if $\mathcal{P}_0...\mathcal{P}_n$ store less than n copies of $\mathcal{D}$ (i.e. one copy), and produce n valid *proofs-of-storage* that convince a verifier $\mathcal{V}$ that $\mathcal{D}$ is stored as n independent copies.

Definition 2.2. (*Outsourcing Attack*) Upon receiving challenge c from verifier $\mathcal{V}$, an attacking prover $\mathcal{A}$ quickly fetches the corresponding $\mathcal{D}$ from another storage provider $\mathcal{P}^*$ and produces the proof, pretending that $\mathcal{A}$ has been storing $\mathcal{D}$ all along.

Definition 2.3. (*Generation Attack*) If attacker $\mathcal{A}$ is in a position to determine $\mathcal{D}$, then $\mathcal{A}$ may choose $\mathcal{D}$ such that $\mathcal{A}$ can re-generate $\mathcal{D}$ on demand. Upon receiving challenge c, $\mathcal{A}$ re-generates $\mathcal{D}$, and produces the proof, pretending that $\mathcal{A}$ has been storing $\mathcal{D}$ all along.

Preventing the *Generation Attack* is difficult, and not usually a problem in the traditional *Cloud Computing* setting, so it has not been a goal of most PoS schemes. However this attack is what prevents most PoS schemes from being used to build *Decentralized Storage Networks*, as an attacker $\mathcal{A}$ could request storage of $\mathcal{D}$ (even pay for it) and then prove storage of $\mathcal{D}$ to collect network rewards. Prior work has attempted to make it irrational to do this by adjusting fee and reward schedules but these approaches prevent useful economic constructions and do not prevent the problem. Completely preventing the *Generation Attack* by forcing such attacker to store the data has been an open problem; PoRep aims solve it.

We now construct a game that tests for the three attacks together and distinguishes PoRep schemes from other PoS schemes. If a PoS scheme passes the game, then it is a PoRep scheme.

Definition 2.4. (RepGame) In the *Replication Game* $\left(\{1,0\} \leftarrow \mathsf{RepGame}(\prod^{\mathsf{PoS}}, \Delta)\right)$, an adversary, $\mathcal{A}$, with a fixed amount of storage l, adaptively interacts with an honest verifier, $\mathcal{V}$, and must prove to $\mathcal{V}$ that $\mathcal{A}$ is storing n *replicas* of data $\mathcal{D}$, such that $n = l + 1$, one more than $\mathcal{A}$ has storage space for. $\mathcal{A}$ chooses data to replicate $\mathcal{D}$, so that $\mathcal{A}$ may generate it on demand. $\mathcal{A}$ can also interact with a separate honest storage provider $\mathcal{P}$ with infinite, free storage; $\mathcal{A}$ may use $\mathcal{P}$ to only store and retrieve arbitrary data, with a latency of Δ. $\mathcal{A}$ may also create and control any Sybil identities that $\mathcal{A}$ wishes. $\mathcal{V}$ and $\mathcal{A}$ use the *Proof-of-Storage* proving scheme $\prod^{\mathsf{PoS}}$. $\mathcal{V}$ asks $\mathcal{A}$ to store n different replicas of $\mathcal{D}$, and runs the setup PoS.Setup for each replica. $\mathcal{A}$ only has enough storage space to store l replicas of $\mathcal{D}$. Then, $\mathcal{V}$ issues a sequence of verification challenges c_i for each replica $i \in \{0...n\}$. $\mathcal{A}$ wins the game if she convinces $\mathcal{V}$ that she is storing all n different replicas, namely if $\mathcal{A}$ produces a set of valid proofs π^{c_i} that convinces $\mathcal{V}$ that $\mathsf{PoS.Verify}(\mathcal{S}_\mathcal{V}, c_i, \pi^{c_i})$ succeeds. If any call to PoS.Verify fails, $\mathcal{A}$ loses.

A PoRep is secure if there is no adversary that wins the PoRep game with more than negligible probability. Armed with these clear attacks and a game than can test for security, we can now formally define PoRep schemes. We use a general scheme construction, similar to the PoS construction above.

Definition 2.5. ($\prod^{\mathsf{PoRep}}$) A general PoRep proving scheme $\prod^{\mathsf{PoRep}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a set of algorithms that together enable a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $\mathcal{P}$ is storing a *replica* $\mathcal{R}^\mathcal{D}$ of data $\mathcal{D}$. No two replicas $\mathcal{R}_i^\mathcal{D}$, $\mathcal{R}_j^\mathcal{D}$ can be deduplicated into the same physical storage; they must be stored independently. The three algorithms are:

- $\mathcal{R}^\mathcal{D}, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{V} \leftarrow \mathsf{PoRep.Setup}(1^\lambda, \mathcal{D})$, where $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{V}$ are scheme-specific setup variables for $\mathcal{P}$ and $\mathcal{V}$ respectively, that depend on the data $\mathcal{D}$, and on a security parameter λ. PoRep.Setup is used to initialize

the proving scheme and give $\mathcal{P}$ and $\mathcal{V}$ information they will use to run PoRep.Prove and PoRep.Verify. Some schemes may require either party to compute PoRep.Setup, require it to be a *secure multi-party computation*, or allow any party to run it.

- $\pi^c \leftarrow \mathsf{PoRep.Prove}(\mathcal{S}_\mathcal{P}, \mathcal{R}^\mathcal{D}, c)$, where c is a challenge, and π^c is a proof that a prover has access to $\mathcal{R}^\mathcal{D}$ a specific *replica* of $\mathcal{D}$. PoRep.Prove is run by $\mathcal{P}$ to produce a π^c for $\mathcal{V}$.
- $\{0,1\} \leftarrow \mathsf{PoRep.Verify}(\mathcal{S}_\mathcal{V}, c, \pi^c)$, which checks whether a proof is correct. PoRep.Verify is run by $\mathcal{V}$ and convinces $\mathcal{V}$ whether $\mathcal{P}$ has been storing $\mathcal{R}^\mathcal{D}$.

A PoRep must be *complete* and *secure*. In addition, PoRep schemes can be designed to have any of the properties described above in Section 1.1.

- (*complete*) if any honest prover $\mathcal{P}$ that stores a replica of $\mathcal{D}$ can always produce valid proofs that convince a verifier $\mathcal{V}$;
- (*secure*) if it can pass RepGame.

2.1 Time Bounded *Proofs-of-Replication*

There are likely to be many different strategies for constructing *Proof-of-Replication* protocols. Any secure construction must prevent the *Sybil Attack*, the *Outsourcing Attack*, and the *Generation Attack* and must pass the RepGame. Some protocols may be able to rely on trusted hardware while others may rely on time bounding. In this work, we are interested in creating constructions that can be deployed to existing systems and do not rely on trusted parties. We give a construction for *time-bounded* PoReps, which can be instantiated using any PDP or PoRet scheme, do not rely on trusted parties, and rely only on local time from the verifier's perspective. This construction is adaptable to *Decentralized Storage Networks*, a *publicly verifiable* setting.

Intuition for preventing the *Sybil Attack*. Ensuring independent physical storage of n copies of $\mathcal{D}$ is similar to ensuring the storage of n different sets of data from which we can derive $\mathcal{D}$. We can treat each independent physical copy differently, force a prover $\mathcal{P}$ to commit to a specific encoding of $\mathcal{D}$ ahead of time, and check $\mathcal{P}$ is storing that specific encoding instead. More formally, we define a *replica* of $\mathcal{D}$ to be an encoding using a per-replica encoding key ek: $\mathcal{R}^\mathcal{D}_{ek} = \mathsf{Encode}(\mathcal{D}, ek)$. Encodings must be distinguishable ($\mathcal{R}^\mathcal{D}_{ek_i} \neq \mathcal{R}^\mathcal{D}_{ek_j}$ when $ek_i \neq ek_j$) and incompressible. In order to recover $\mathcal{D}$, the encoding must be reversible: $\mathcal{D} = \mathsf{Decode}(\mathcal{R}^\mathcal{D}_{ek}, ek)$. Creating n different replicas, each with encoding key ek_i for replica $i \in 0...n$, forces any number of Sybil identities or colluding participants to prove each of those n replicas existed when producing a valid proof.

Intuition for preventing the *Outsourcing* and *Generation Attacks* We must still ensure that provers cannot get the replica just-in-time (between receiving the challenge c and producing the proof π^c), either by retrieving it from outsourced storage or by producing it by encoding $\mathcal{D}$. To achieve this, we can simply make attackers be distinguishably slower than an honest prover responding to a challenge. Computing $\mathsf{Encode}(\mathcal{D}, ek)$ must take a distinguishable amount of time, such that a verifier, $\mathcal{V}$, can distinguish between:

(a) $\mathcal{T}^{\mathsf{honest}} = \mathsf{RTT}^{\mathcal{V}\rightarrow\mathcal{P}\rightarrow\mathcal{V}} + \mathsf{Time}(\mathsf{PoRep.Prove}(\mathcal{S}_\mathcal{P}, \mathcal{R}^\mathcal{D}_{ek}, c))$
(b) $\mathcal{T}^{\mathsf{attack}} = \mathsf{RTT}^{\mathcal{V}\rightarrow\mathcal{P}\rightarrow\mathcal{V}} + \mathsf{Time}(\mathsf{PoRep.Prove}(\mathcal{S}_\mathcal{P}, \mathsf{Encode}(\mathcal{D}, ek), c))$

where $\mathsf{RTT}^{\mathcal{V}\rightarrow\mathcal{P}\rightarrow\mathcal{V}}$ is the round-trip time from $\mathcal{V}$ to $\mathcal{P}$ and back to $\mathcal{V}$. For an attacker running Encode just-in-time to be distinguishable from a slow or unlucky but honest prover, computing Encode must run distinguishably slower than acceptable variance in $\mathsf{RTT}^{\mathcal{V}\rightarrow\mathcal{P}\rightarrow\mathcal{V}}$ and $\mathsf{PoRep.Prove}(\mathcal{S}_\mathcal{P}, \mathcal{R}^\mathcal{D}_{ek}, c)$. From the perspective of $\mathcal{V}$, the wall clock running time of $\mathcal{P}$ computing $\mathsf{Encode}(\mathcal{D}, ek)$ must be noticeable. Statistical estimation of RTT and PoRep.Prove to determine the acceptable variance can give a lower bound on the amount of time required. This may be required in systems where variance in RTT and PoRep.Prove do not dominate the time required for the proving step, but rather their minimums do (e.g. proofs across the interplanetary internet, or collocated in a datacenter). For distributed systems across Earth, we assume

variance in RTT to dominate, and to establish a significant bound we set the minimum time of Encode $\mathcal{T}^{\mathsf{Encode}} = 10 \times \mathcal{T}^{\mathsf{honest}}$ or $100 \times \mathcal{T}^{\mathsf{honest}}$. As long as verifiers can clearly distinguish $\mathcal{T}^{\mathsf{honest}} << \mathcal{T}^{\mathsf{attack}}$ we defeat both the *Outsourcing* and *Generation Attacks*.

Intuitions for Encode. The remaining question is what function should Encode be? We have explored some of the requirements: (a) it must be slow enough to distinguish $\mathcal{T}^{\mathsf{honest}} << \mathcal{T}^{\mathsf{attack}}$; (b) it must be reversible; (c) the output should be determined from an encoding key; and (d) information should not be compressible across replicas. Additional design goals follow: (a) Decode should allow fast recovery of $\mathcal{D}$, ideally the running time of Decode grows sub-linearly in respect to Encode; (b) Encode's running time should scale arbitrarily with a tunable parameter; (c) Encode should be publicly verifiable, so that anybody, including $\mathcal{P}$, can verify it; (d) Encode should use all of $\mathcal{D}$ multiple times before finishing, so that a partial replica cannot be computed from partial $\mathcal{D}$, which would affect timing assumptions; (e) Encode should support high-entropy encoding keys; and (f) a slight variation (single bit flip) in the encoding key should change the encoding completely. What we need is a PRP (*pseudo-random permutation*) that matches our desires.

Slowable PRPs**.** It is easy to construct a PRP that can be slowed down arbitrarily by using a *block cipher* (BC) in *cipher block chaining* (CBC) mode. The following three variations of CBC sequentially increase in complexity to achieve the desired properties:

- (*streaming*) We simply run a modified version of CBC.Encrypt that encrypts each block τ times before moving on to the next. This slows down the encryption by a factor of τ, making it run in $\mathcal{O}(n\tau)$ time, where n is the size of the input data. Note that encryption is still sequential; this is important to slow down encryption cheaply, in that decryption is still parallelizable ($\mathcal{O}(n\tau)$ time, with up to τ parallelization), which speeds up recovery of the original data and allows random reads. The only issue with this scheme is that it is *depth-first*: it computes and outputs each cipher text block before reading the next input block (i.e. streaming process). This makes it possible to encrypt having only a partial input file, and to start using the encrypted output after each iteration, significantly lowering the slow-down.
- (*shuffling*). To fix the issues with the *streaming* scheme above we can try some random shuffling: before and after starting CBC.Encrypt, apply a random shuffle that is likely to disperse information across all the blocks, making it impossible to use sequential segments on an encrypted block until the whole data is encrypted. This also affects decryption, as a decryptor must have the entire file and de-scramble it before running the streaming and parallelizable CBC.Decrypt.
- (*layering*) Another version runs all of CBC.Encrypt for τ iterations, taking the output of the last iteration as the input of the next, making sure to chain across iterations. This chaining preserves the sequential property of CBC encryption: the last cipher block of iteration $\tau - 1$ is chained with the first block of iteration τ. This sequentiality ensures the encryption cannot be parallelized. The running time of CBC.Encrypt is still $\mathcal{O}(n\tau)$ and CBC.Decrypt is still parallelizable ($\mathcal{O}(n\tau)$ with up to τ parallelization), but requires decrypting in whole layers before recovering any plaintext.

We can make any of these constructions *publicly verifiable* without the data (cheaply) by computing them within a SCIP scheme (SNARKs, STARKs) [7, 4, 3, 2]. Doing this would be expensive, and it is likely primitives used in SCIP scheme constructions can be used directly in a cheaper way. We believe this is an open problem.

Definition 2.6. (Seal) We define a pair of encoding functions Seal and Unseal, inverses of each other, and parametrized over any secure *block cipher* algorithm BC and number of rounds t:

$$\mathsf{Seal}^{\tau}_{\mathsf{BC}}(ek, \mathcal{D}) := \mathsf{BC.CBC.Encrypt}^{\tau}(ek, \mathcal{D})$$
$$\mathsf{Unseal}^{\tau}_{\mathsf{BC}}(ek, \mathcal{R}^{\mathcal{D}}_{ek}) := \mathsf{BC.CBC.Decrypt}^{\tau}(ek, \mathcal{R}^{\mathcal{D}}_{ek})$$

where BC.CBC.{Encrypt, Decrypt} are the Encrypt,Decrypt functions of BC in CBC mode; τ is the number of encryption rounds to perform, which is chosen to slow down Seal; and $\{\mathsf{Encrypt}^{\tau}, \mathsf{Decrypt}^{\tau}\}$ is the recursive application of Encrypt or Decrypt for τ iterations, where the last cipher block of each iteration is chained back onto the first block of the following iteration, ensuring that the entire encryption is sequential.

With $\mathsf{Seal}^{\tau}_{\mathsf{BC}}$ and any existing PDP or PoRet scheme, we can construct a *time bounded* PoRep scheme, where the proofs prove that the storage of a specific *replica* is encoded with $\mathsf{Seal}^{\tau}_{\mathsf{BC}}$. The scheme is *time bounded* because the proofs only prove possession of the *replica* at the time of the challenge if they are produced by $\mathcal{P}$ and checked by $\mathcal{V}$ in less time than the running time of $\mathsf{Seal}^{\tau}_{\mathsf{BC}}$.

Definition 2.7. ($\prod^{\mathsf{SealPoRep}}$) A *time-bounded* Seal-based PoRep proving scheme $\prod^{\mathsf{SealPoRep}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a set of algorithms that together enable a prover $\mathcal{P}$ to convince a verifier $\mathcal{V}$ that $\mathcal{P}$ is storing independent *replica* $\mathcal{R}^{\mathcal{D}}_{ek}$ of data $\mathcal{D}$, where $\mathcal{R}^{\mathcal{D}}_{ek} = \mathsf{Seal}^{\tau}_{\mathsf{BC}}(ek, \mathcal{D})$ for some encoding key ek, a secure block cipher BC, and a timing parameter τ. These algorithms rely on another *Proof-of-Storage* proving scheme $\prod^{\mathsf{PoS}}$, which is used internally to prove storage of the replica, and may be either a PDP or PoRet scheme. The three algorithms are:

- $\mathcal{R}^{\mathcal{D}}_{ek}, ek \leftarrow \mathsf{SealPoRep.Setup}(1^{\lambda}, \tau, \mathcal{D})$ where ek is an encoding key unique to a single setup. ek may be chosen randomly, or derived from the identity of $\mathcal{P}$ and a unique replica number. SealPoRep.Setup is used to initialize the proving scheme: to choose an encoding key ek, to compute $\mathcal{R}^{\mathcal{D}}_{ek} = \mathsf{Seal}^{\tau}_{\mathsf{BC}}(ek, \mathcal{D})$, and to compute $\mathsf{PoS.Setup}(1^{\lambda}, \mathcal{R}^{\mathcal{D}}_{ek})$, all of which $\mathcal{P}$ and $\mathcal{V}$ will use to run SealPoRep.Prove and SealPoRep.Verify. With a *publicly verifiable* $\mathsf{Seal}^{\tau}_{\mathsf{BC}}$, any party can run the setup, even $\mathcal{P}$. The τ timing parameter must be chosen such that running $\mathsf{Seal}^{\tau}_{\mathsf{BC}}$ is distinguishibly more expensive than just proving and verifying.
- $\pi^c \leftarrow \mathsf{SealPoRep.Prove}(\mathcal{R}^{\mathcal{D}}_{ek}, c)$ where c is a challenge, and π^c is a proof that a prover has access to $\mathcal{R}^{\mathcal{D}}_{ek}$. SealPoRep.Prove computes $\pi^c = \mathsf{PoS.Prove}(\mathcal{R}^{\mathcal{D}}_{ek}, c)$. It is run by $\mathcal{P}$.
- $\{0,1\} \leftarrow \mathsf{SealPoRep.Verify}(c, \pi^c)$ which checks whether a proof is correct. SealPoRep.Verify internally computes $\{0,1\} \leftarrow \mathsf{PoS.Verify}(c, \pi^c)$. It is run by $\mathcal{V}$, and convinces $\mathcal{V}$ whether $\mathcal{P}$ has been storing $\mathcal{R}^{\mathcal{D}}_{ek}$

3 *Proofs-of-Spacetime*

Usually, most PoS and PoSpace schemes are described in terms of interactive challenge settings, where a verifier $\mathcal{V}$ issues single or periodic challenges to a prover $\mathcal{P}$. Unpredictable yet frequent challenges can give $\mathcal{V}$ confidence that $\mathcal{P}$ has been correctly storing the required data $\mathcal{D}$ for the duration of time challenged. These challenges must be frequent and unpredictable, as too infrequent or predictable challenges could give $\mathcal{P}$ a chance to cheat by retrieving $\mathcal{D}$ (from other outsourced storage or from a secondary market) in advance of $\mathcal{V}$'s challenge. $\mathcal{V}$'s confidence increases with the frequency of challenges. This is a useful way for a *Cloud Storage* client ($\mathcal{V}$) to ensure a particular service ($\mathcal{P}$) is correctly storing the client's data over time. However, these interactive checks require the clients to be online and spend bandwidth and computation resources during all challenges.

Time-bounded PoReps could allow the time intervals to be large, as the Seal function can be scaled to be significantly larger than the maximum desired time between intervals. Yet, this would not obviate the need for periodic challenges; many challenges would still be needed over long periods of time, and these would require $\mathcal{V}$ to be online and spending resources to perform them. In both *Cloud Storage* and *Decentralized Storage Network* settings, it would be quite useful to allow clients to check proofs infrequently, perhaps coupled with other necessary accesses, such as retrieving $\mathcal{D}$. It would also be useful to have an auditable record that shows $\mathcal{P}$ has been behaving correctly – storing $\mathcal{D}$ – for the entire duration of time. This record could even be time-stamped into public ledgers such as blockchains.

Chaining Sequential Proofs. We can construct a sequence of challenges and proofs where the challenge at iteration n (c_n) is derived deterministically from the proof at iteration $n-1$ (π_{n-1}). Let a *proof-chain* be these sequential challenges and proofs stored together, for a verifier to inspect all at once. Note that every proof must be correctly produced; if any proof fails to verify, the whole chain fails to verify.

Definition 3.1. (*proof-chain*) A *proof-chain* is a verifiable data-structure that chains together a sequence of challenges and proofs. For iteration n, let c_n be a challenge, π_n be a proof for c_n, and $\mathcal{C}_n$ be the *proof-chain* formed by extending $\mathcal{C}_{n-1}$ with π_n. Given a proof scheme with functions (Prove, Verify), randomness r, and a *collision resistant hash-function* (CRH) $\mathcal{H}$; we can construct the data structure and verify it with

VerifyChain, according to the following rules:

- $c_0 \leftarrow \mathcal{H}(r)$
- $c_n \leftarrow \mathcal{H}(n||\pi_{n-1})$
- $\pi_n \leftarrow \mathsf{Prove}(c_n)$
- $\mathcal{C}_0 \leftarrow (c_0, \pi_0)$
- $\mathcal{C}_n \leftarrow (\mathcal{C}_{n-1}, \pi_n)$
- $\{0,1\} \leftarrow \mathsf{VerifyChain}(\mathcal{C}_0) = \mathsf{Verify}(c_0, \pi_0)$
- $\{0,1\} \leftarrow \mathsf{VerifyChain}(\mathcal{C}_n) = \mathsf{VerifyChain}(\mathcal{C}_{n-1})$ & $\mathsf{Verify}(c_n, \pi_n)$

Time and *proof-chains*. Such a *proof-chain* would be a verifiable record of a prover producing proofs over a duration of time. There are schemes with predictable computation time, where a verifier $\mathcal{V}$ can know how long it takes a prover $\mathcal{P}$ to produce each iteration (i.e. with some well-known and bounded variance). Using such schemes, $\mathcal{V}$ could ask $\mathcal{P}$ to begin producing a chain at time t_0. At a future time t_n, $\mathcal{V}$ can ask $\mathcal{P}$ to return the chain, which should contain as many entries as can be computed within the interval $t_n \rightarrow t_0$. If the chain passes verification, $\mathcal{V}$ knows that $\mathcal{P}$ has spent that interval of time $(t_0 \rightarrow t_n)$ producing the proof chain. This also implies $\mathcal{P}$ has been storing $\mathcal{D}$, or at least has had sufficiently fast access to $\mathcal{D}$, during the entire interval. $\mathcal{V}$ can have the same degree of confidence as with a sequence of online PoS challenges. Since producing the *proof-chain* is a sequential process, with parallelization only possible in the proof algorithm itself, $\mathcal{P}$ need only provide enough processing power to compute a single proof at a time, and $\mathcal{P}$ cannot spend more computation resources in parallel to speed up the process and cheat the expectations. The nth iteration of the *proof-chain* has similar guarantees as an equivalent online PoS challenge.

Reducing disk accesses and time variance. The process of computing such a *proof-chain* would force a prover $\mathcal{P}$ to be constantly running a PoS proving process, which may require significant computation resources and disk accesses. This should not be a task too onerous for most modern computers to perform. However, some use-cases may need to reduce the amount of disk accesses, or to reduce the variance in the time required to compute each iteration of the *proof-chain*. To do so, we can adjust the chaining scheme to run some additional *sequential* computation when deriving the next challenge. This computation can be tuned to have the desired properties: low time variance, no disk accesses, CPU bounded, memory bounded, storage bounded, etc. For example, the Sloth hash function [9] is arbitrarily scalable in time, is CPU bounded, has low time variance, and makes zero disk accesses. The Balloon hash function [5] is arbitrarily scalable in time, is memory bounded, has low time variance, and makes zero disk accesses. These are only two examples of sequential processes with predictable running times can be used to tune the properties of the *proof-chain* computation.

***Proof-chains* and spacetime.** The creation of this sequential *proof-chain* of PoS or PoSpace proofs implies that a specific amount of space was kept occupied storing data $\mathcal{D}$ for a specific duration of time. In other words, a slice of *spacetime* was committed to storing the relevant data $\mathcal{D}$. Hence we call this chaining of sequential proofs a *Proof-of-Spacetime* (PoSt). These proofs are inherently *time-bounded*, in that a PoSt proof for a duration of time t is only valid if verified shortly after, but the chain can be extended arbitrarily. The only bound is the variance in the time to compute each iteration. The variance of computing the whole PoSt *proof-chains* must be low enough to have confidence over the time interval.

PoSt to the blockchain. The sequential PoSt *proof-chains* can get arbitrarily long when involving time-stamping services – such as the Bitcoin blockchain. Given a time-stamping service trusted by both $\mathcal{P}$ and $\mathcal{V}$, $\mathcal{P}$ can periodically time-stamp the head of the *proof-chain* (creating a checkpoint), effectively anchoring the *proof-chain* in time. This can happen while $\mathcal{V}$ is offline. In the future, $\mathcal{V}$ can decide to audit the *proof-chain* and its time-stamped checkpoints, and verify that $\mathcal{P}$ correctly produced the *proof-chain* during the right intervals of time.

Definition 3.2. ($\prod^{\mathsf{PoSt}}$) A general PoSt proving scheme $\prod^{\mathsf{PoSt}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is a set of algorithms that together enable a prover $\mathcal{P}$ to produce an incremental *time-bounded proof-chain* $\mathcal{C}_n$ which proves to a verifier $\mathcal{V}$ that $\mathcal{P}$ has been storing data $\mathcal{D}$ during iterations $0 \rightarrow n$ of the *proof-chain*. The *proof-chain* must be periodically checked or time-stamped. These algorithms rely on a *Proof-of-Space* (PoSpace) proving scheme $\prod^{\mathsf{PoSpace}}$, which may also be a *Proof-of-Storage* scheme ($\prod^{\mathsf{PoS}}$). The three algorithms are:

- $\mathcal{D}, c_0, \mathcal{S}_\mathcal{P}, \mathcal{S}_\mathcal{V} \leftarrow \mathsf{PoSt.Setup}(1^\lambda, \mathcal{D})$ where c_0 is an initial random challenge not controlled by $\mathcal{P}$, $\mathcal{S}_\mathcal{P}$ and $\mathcal{S}_\mathcal{V}$ are scheme-specific setup variables for $\mathcal{P}$ and $\mathcal{V}$ respectively, that depend on the data $\mathcal{D}$, and on a security parameter λ. PoSt.Setup is used to initialize the proving scheme and give $\mathcal{P}$ and $\mathcal{V}$ information they will use to run PoSt.Prove and PoSt.Verify. Some schemes may require either party to compute PoSt.Setup, require it to be a *secure multi-party computation*, or allow any party to run it.
- $\mathcal{C}_{n+1} \leftarrow \mathsf{PoSt.Prove}(\mathcal{S}_\mathcal{P}, \mathcal{D}, \mathcal{C}_n)$ where $\mathcal{C}_n$ and $\mathcal{C}_{n+1}$ are *proof-chains* that prove $\mathcal{P}$ had access to $\mathcal{D}$ up to iterations n and $n+1$ respectively. PoSt.Prove is run by $\mathcal{P}$ to produce each *proof-chain* iteration, extending the chain with the next proof (π_{n+1}), as computed by PoSpace.Prove. Note that the challenges and proofs are computed according to the proving scheme and the *proof-chain* construction. The *proof-chain* and its proofs will later be checked by $\mathcal{V}$.
- $\{0, 1\} \leftarrow \mathsf{PoSt.Verify}(\mathcal{S}_\mathcal{V}, \mathcal{C}_n)$ which checks whether *proof-chain* is correct, by verifying every iteration's proof with PoSpace.Verify. PoSt.Verify is run by $\mathcal{V}$ and convinces $\mathcal{V}$ whether $\mathcal{P}$ has been storing $\mathcal{D}$ during iterations $0 \rightarrow n$.

4 Realizable Constructions (TODO)

4.1 Realizable time-bounded *Proof-of-Replication*

SealPoRep with $\mathsf{Seal}^\tau_{\mathsf{AES}-256}$, and τ such that $\mathsf{Seal}^\tau_{\mathsf{AES}-256}$ takes 10-100x the honest prover/verifier time.

4.2 Realizable useful *Proofs-of-Spacetime*

A PoSt with the PoRep from 5.1, checked or time-stamped frequently (into a blockchain).

Acknowledgements

This work is the cumulative effort of multiple individuals within the Protocol Labs team, and would not have been possible without their the help, comments, and review.

References

[1] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 598–609, New York, NY, USA, 2007. ACM.

[2] E. Ben-Sasson, I. Bentov, A. Chiesa, A. Gabizon, D. Genkin, M. Hamilis, E. Pergament, M. Riabzev, M. Silberstein, E. Tromer, et al. Computational integrity with a public random string from quasi-linear pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 551–579. Springer, 2017.

[3] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *Advances in Cryptology–CRYPTO 2013*, pages 90–108. Springer, 2013.

[4] N. Bitansky, A. Chiesa, and Y. Ishai. Succinct non-interactive arguments via linear interactive proofs. Springer, 2013.

[5] D. Boneh, H. Corrigan-Gibbs, and S. Schechter. Balloon hashing: A memory-hard function providing provable protection against sequential attacks. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 220–248. Springer, 2016.

[6] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 139–147, London, UK, UK, 1993. Springer-Verlag.

[7] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[8] A. Juels and B. S. Kaliski, Jr. Pors: Proofs of retrievability for large files. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 584–597, New York, NY, USA, 2007. ACM.

[9] A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx.

[10] T. Moran and I. Orlov. Proofs of space-time and rational proofs of storage. Cryptology ePrint Archive, Report 2016/035, 2016. http://eprint.iacr.org/2016/035.

[11] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[12] H. Shacham and B. Waters. Compact proofs of retrievability. In *Proceedings of the 14th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '08, pages 90–107, Berlin, Heidelberg, 2008. Springer-Verlag.

IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)

Juan Benet
juan@benet.ai

ABSTRACT

The InterPlanetary File System (IPFS) is a peer-to-peer distributed file system that seeks to connect all computing devices with the same system of files. In some ways, IPFS is similar to the Web, but IPFS could be seen as a single BitTorrent swarm, exchanging objects within one Git repository. In other words, IPFS provides a high throughput content-addressed block storage model, with content-addressed hyper links. This forms a generalized Merkle DAG, a data structure upon which one can build versioned file systems, blockchains, and even a Permanent Web. IPFS combines a distributed hashtable, an incentivized block exchange, and a self-certifying namespace. IPFS has no single point of failure, and nodes do not need to trust each other.

1. INTRODUCTION

There have been many attempts at constructing a global distributed file system. Some systems have seen significant success, and others failed completely. Among the academic attempts, AFS [6] has succeeded widely and is still in use today. Others [7, **?**] have not attained the same success. Outside of academia, the most successful systems have been peer-to-peer file-sharing applications primarily geared toward large media (audio and video). Most notably, Napster, KaZaA, and BitTorrent [2] deployed large file distribution systems supporting over 100 million simultaneous users. Even today, BitTorrent maintains a massive deployment where tens of millions of nodes churn daily [16]. These applications saw greater numbers of users and files distributed than their academic file system counterparts. However, the applications were not designed as infrastructure to be built upon. While there have been successful repurposings[1], no general file-system has emerged that offers global, low-latency, and decentralized distribution.

Perhaps this is because a "good enough" system for most use cases already exists: HTTP. By far, HTTP is the most successful "distributed system of files" ever deployed. Coupled with the browser, HTTP has had enormous technical and social impact. It has become the de facto way to transmit files across the internet. Yet, it fails to take advantage of dozens of brilliant file distribution techniques invented in the last fifteen years. From one prespective, evolving Web infrastructure is near-impossible, given the number of backwards compatibility constraints and the number of strong parties invested in the current model. But from another perspective, new protocols have emerged and gained wide use since the emergence of HTTP. What is lacking is upgrading design: enhancing the current HTTP web, and introducing new functionality without degrading user experience.

Industry has gotten away with using HTTP this long because moving small files around is relatively cheap, even for small organizations with lots of traffic. But we are entering a new era of data distribution with new challenges: (a) hosting and distributing petabyte datasets, (b) computing on large data across organizations, (c) high-volume high-definition on-demand or real-time media streams, (d) versioning and linking of massive datasets, (e) preventing accidental disappearance of important files, and more. Many of these can be boiled down to "lots of data, accessible everywhere." Pressed by critical features and bandwidth concerns, we have already given up HTTP for different data distribution protocols. The next step is making them part of the Web itself.

Orthogonal to efficient data distribution, version control systems have managed to develop important data collaboration workflows. Git, the distributed source code version control system, developed many useful ways to model and implement distributed data operations. The Git toolchain offers versatile versioning functionality that large file distribution systems severely lack. New solutions inspired by Git are emerging, such as Camlistore [**?**], a personal file storage system, and Dat [**?**] a data collaboration toolchain and dataset package manager. Git has already influenced distributed filesystem design [9], as its content addressed Merkle DAG data model enables powerful file distribution strategies. What remains to be explored is how this data structure can influence the design of high-throughput oriented file systems, and how it might upgrade the Web itself.

This paper introduces IPFS, a novel peer-to-peer version-controlled filesystem seeking to reconcile these issues. IPFS synthesizes learnings from many past successful systems. Careful interface-focused integration yields a system greater than the sum of its parts. The central IPFS principle is modeling *all data* as part of the same Merkle DAG.

2. BACKGROUND

This section reviews important properties of successful peer-to-peer systems, which IPFS combines.

2.1 Distributed Hash Tables

Distributed Hash Tables (DHTs) are widely used to coordinate and maintain metadata about peer-to-peer systems.

[1]For example, Linux distributions use BitTorrent to transmit disk images, and Blizzard, Inc. uses it to distribute video game content.

For example, the BitTorrent MainlineDHT tracks sets of peers part of a torrent swarm.

2.1.1 Kademlia DHT

Kademlia [10] is a popular DHT that provides:

1. Efficient lookup through massive networks: queries on average contact $\lceil log_2(n) \rceil$ nodes. (e.g. 20 hops for a network of $10,000,000$ nodes).
2. Low coordination overhead: it optimizes the number of control messages it sends to other nodes.
3. Resistance to various attacks by preferring long-lived nodes.
4. Wide usage in peer-to-peer applications, including Gnutella and BitTorrent, forming networks of over 20 million nodes [16].

2.1.2 Coral DSHT

While some peer-to-peer filesystems store data blocks directly in DHTs, this "wastes storage and bandwidth, as data must be stored at nodes where it is not needed" [5]. The Coral DSHT extends Kademlia in three particularly important ways:

1. Kademlia stores values in nodes whose ids are "nearest" (using XOR-distance) to the key. This does not take into account application data locality, ignores "far" nodes that may already have the data, and forces "nearest" nodes to store it, whether they need it or not. This wastes significant storage and bandwith. Instead, Coral stores addresses to peers who can provide the data blocks.
2. Coral relaxes the DHT API from `get_value(key)` to `get_any_values(key)` (the "sloppy" in DSHT). This still works since Coral users only need a single (working) peer, not the complete list. In return, Coral can distribute only subsets of the values to the "nearest" nodes, avoiding hot-spots (overloading *all the nearest nodes* when a key becomes popular).
3. Additionally, Coral organizes a hierarchy of separate DSHTs called *clusters* depending on region and size. This enables nodes to query peers in their region first, "finding nearby data without querying distant nodes" [5] and greatly reducing the latency of lookups.

2.1.3 S/Kademlia DHT

S/Kademlia [1] extends Kademlia to protect against malicious attacks in two particularly important ways:

1. S/Kademlia provides schemes to secure `NodeId` generation, and prevent Sybill attacks. It requires nodes to create a PKI key pair, derive their identity from it, and sign their messages to each other. One scheme includes a proof-of-work crypto puzzle to make generating Sybills expensive.
2. S/Kademlia nodes lookup values over disjoint paths, in order to ensure honest nodes can connect to each other in the presence of a large fraction of adversaries in the network. S/Kademlia achieves a success rate of 0.85 even with an adversarial fraction as large as half of the nodes.

2.2 Block Exchanges - BitTorrent

BitTorrent [3] is a widely successful peer-to-peer filesharing system, which succeeds in coordinating networks of untrusting peers (swarms) to cooperate in distributing pieces of files to each other. Key features from BitTorrent and its ecosystem that inform IPFS design include:

1. BitTorrent's data exchange protocol uses a quasi tit-for-tat strategy that rewards nodes who contribute to each other, and punishes nodes who only leech others' resources.
2. BitTorrent peers track the availability of file pieces, prioritizing sending rarest pieces first. This takes load off seeds, making non-seed peers capable of trading with each other.
3. BitTorrent's standard tit-for-tat is vulnerable to some exploitative bandwidth sharing strategies. PropShare [8] is a different peer bandwidth allocation strategy that better resists exploitative strategies, and improves the performance of swarms.

2.3 Version Control Systems - Git

Version Control Systems provide facilities to model files changing over time and distribute different versions efficiently. The popular version control system Git provides a powerful Merkle DAG [2] object model that captures changes to a filesystem tree in a distributed-friendly way.

1. Immutable objects represent Files (`blob`), Directories (`tree`), and Changes (`commit`).
2. Objects are content-addressed, by the cryptographic hash of their contents.
3. Links to other objects are embedded, forming a Merkle DAG. This provides many useful integrity and workflow properties.
4. Most versioning metadata (branches, tags, etc.) are simply pointer references, and thus inexpensive to create and update.
5. Version changes only update references or add objects.
6. Distributing version changes to other users is simply transferring objects and updating remote references.

2.4 Self-Certified Filesystems - SFS

SFS [12, 11] proposed compelling implementations of both (a) distributed trust chains, and (b) egalitarian shared global namespaces. SFS introduced a technique for building *Self-Certified Filesystems*: addressing remote filesystems using the following scheme

```
/sfs/<Location>:<HostID>
```

where `Location` is the server network address, and:

```
HostID = hash(public_key || Location)
```

Thus the *name* of an SFS file system certifies its server. The user can verify the public key offered by the server, negotiate a shared secret, and secure all traffic. All SFS instances share a global namespace where name allocation is cryptographic, not gated by any centralized body.

[2] Merkle Directed Acyclic Graph – similar but more general construction than a Merkle Tree. Deduplicated, does not need to be balanced, and non-leaf nodes contain data.

3. IPFS DESIGN

IPFS is a distributed file system which synthesizes successful ideas from previous peer-to-peer sytems, including DHTs, BitTorrent, Git, and SFS. The contribution of IPFS is simplifying, evolving, and connecting proven techniques into a single cohesive system, greater than the sum of its parts. IPFS presents a new platform for writing and deploying applications, and a new system for distributing and versioning large data. IPFS could even evolve the web itself.

IPFS is peer-to-peer; no nodes are privileged. IPFS nodes store IPFS objects in local storage. Nodes connect to each other and transfer objects. These objects represent files and other data structures. The IPFS Protocol is divided into a stack of sub-protocols responsible for different functionality:

1. **Identities** - manage node identity generation and verification. Described in Section 3.1.
2. **Network** - manages connections to other peers, uses various underlying network protocols. Configurable. Described in Section 3.2.
3. **Routing** - maintains information to locate specific peers and objects. Responds to both local and remote queries. Defaults to a DHT, but is swappable. Described in Section 3.3.
4. **Exchange** - a novel block exchange protocol (BitSwap) that governs efficient block distribution. Modelled as a market, weakly incentivizes data replication. Trade Strategies swappable. Described in Section 3.4.
5. **Objects** - a Merkle DAG of content-addressed immutable objects with links. Used to represent arbitrary datastructures, e.g. file hierarchies and communication systems. Described in Section 3.5.
6. **Files** - versioned file system hierarchy inspired by Git. Described in Section 3.6.
7. **Naming** - A self-certifying mutable name system. Described in Section 3.7.

These subsystems are not independent; they are integrated and leverage blended properties. However, it is useful to describe them separately, building the protocol stack from the bottom up.

Notation: data structures and functions below are specified in Go syntax.

3.1 Identities

Nodes are identified by a `NodeId`, the cryptographic hash[3] of a public-key, created with S/Kademlia's static crypto puzzle [1]. Nodes store their public and private keys (encrypted with a passphrase). Users are free to instatiate a "new" node identity on every launch, though that loses accrued network benefits. Nodes are incentivized to remain the same.

```
type NodeId Multihash
type Multihash []byte
// self-describing cryptographic hash digest

type PublicKey []byte
type PrivateKey []byte
// self-describing keys

type Node struct {
  NodeId NodeID
  PubKey PublicKey
  PriKey PrivateKey
}
```

S/Kademlia based IPFS identity generation:

```
difficulty = <integer parameter>
n = Node{}
do {
  n.PubKey, n.PrivKey = PKI.genKeyPair()
  n.NodeId = hash(hash(n.PubKey))
  p = count_preceding_zero_bits(n.NodeId)
} while (p < difficulty)
```

Upon first connecting, peers exchange public keys, and check: `hash(other.PublicKey) equals other.NodeId`. If not, the connection is terminated.

Note on Cryptographic Functions.

Rather than locking the system to a particular set of function choices, IPFS favors self-describing values. Hash digest values are stored in `multihash` format, which includes a short header specifying the hash function used, and the digest length in bytes. Example:

```
<function code><digest length><digest bytes>
```

This allows the system to (a) choose the best function for the use case (e.g. stronger security vs faster performance), and (b) evolve as function choices change. Self-describing values allow using different parameter choices compatibly.

3.2 Network

IPFS nodes communicate regualarly with hundreds of other nodes in the network, potentially across the wide internet. The IPFS network stack features:

- **Transport:** IPFS can use any transport protocol, and is best suited for WebRTC DataChannels [**?**] (for browser connectivity) or uTP(LEDBAT [14]).
- **Reliability:** IPFS can provide reliability if underlying networks do not provide it, using uTP (LEDBAT [14]) or SCTP [15].
- **Connectivity:** IPFS also uses the ICE NAT traversal techniques [13].
- **Integrity:** optionally checks integrity of messages using a hash checksum.
- **Authenticity:** optionally checks authenticity of messages using HMAC with sender's public key.

3.2.1 Note on Peer Addressing

IPFS can use any network; it does not rely on or assume access to IP. This allows IPFS to be used in overlay networks. IPFS stores addresses as `multiaddr` formatted byte strings for the underlying network to use. `multiaddr` provides a way to express addresses and their protocols, including support for encapsulation. For example:

[3]Throughout this document, *hash* and *checksum* refer specifically to cryptographic hash checksums of data.

```
# an SCTP/IPv4 connection
/ip4/10.20.30.40/sctp/1234/

# an SCTP/IPv4 connection proxied over TCP/IPv4
/ip4/5.6.7.8/tcp/5678/ip4/1.2.3.4/sctp/1234/
```

3.3 Routing

IPFS nodes require a routing system that can find (a) other peers' network addresses and (b) peers who can serve particular objects. IPFS achieves this using a DSHT based on S/Kademlia and Coral, using the properties discussed in 2.1. The size of objects and use patterns of IPFS are similar to Coral [5] and Mainline [16], so the IPFS DHT makes a distinction for values stored based on their size. Small values (equal to or less than `1KB`) are stored directly on the DHT. For values larger, the DHT stores references, which are the `NodeIds` of peers who can serve the block.

The interface of this DSHT is the following:

```
type IPFSRouting interface {

  FindPeer(node NodeId)
  // gets a particular peer's network address

  SetValue(key []bytes, value []bytes)
  // stores a small metadata value in DHT

  GetValue(key []bytes)
  // retrieves small metadata value from DHT

  ProvideValue(key Multihash)
  // announces this node can serve a large value

  FindValuePeers(key Multihash, min int)
  // gets a number of peers serving a large value
}
```

Note: different use cases will call for substantially different routing systems (e.g. DHT in wide network, static HT in local network). Thus the IPFS routing system can be swapped for one that fits users' needs. As long as the interface above is met, the rest of the system will continue to function.

3.4 Block Exchange - BitSwap Protocol

In IPFS, data distribution happens by exchanging blocks with peers using a BitTorrent inspired protocol: BitSwap. Like BitTorrent, BitSwap peers are looking to acquire a set of blocks (`want_list`), and have another set of blocks to offer in exchange (`have_list`). Unlike BitTorrent, BitSwap is not limited to the blocks in one torrent. BitSwap operates as a persistent marketplace where node can acquire the blocks they need, regardless of what files those blocks are part of. The blocks could come from completely unrelated files in the filesystem. Nodes come together to barter in the marketplace.

While the notion of a barter system implies a virtual currency could be created, this would require a global ledger to track ownership and transfer of the currency. This can be implemented as a BitSwap Strategy, and will be explored in a future paper.

In the base case, BitSwap nodes have to provide direct value to each other in the form of blocks. This works fine when the distribution of blocks across nodes is complementary, meaning they have what the other wants. Often, this will not be the case. In some cases, nodes must *work* for their blocks. In the case that a node has nothing that its peers want (or nothing at all), it seeks the pieces its peers want, with lower priority than what the node wants itself. This incentivizes nodes to cache and disseminate rare pieces, even if they are not interested in them directly.

3.4.1 BitSwap Credit

The protocol must also incentivize nodes to seed when they do not need anything in particular, as they might have the blocks others want. Thus, BitSwap nodes send blocks to their peers optimistically, expecting the debt to be repaid. But leeches (free-loading nodes that never share) must be protected against. A simple credit-like system solves the problem:

1. Peers track their balance (in bytes verified) with other nodes.
2. Peers send blocks to debtor peers probabilistically, according to a function that falls as debt increases.

Note that if a node decides not to send to a peer, the node subsequently ignores the peer for an `ignore_cooldown` timeout. This prevents senders from trying to game the probability by just causing more dice-rolls. (Default BitSwap is 10 seconds).

3.4.2 BitSwap Strategy

The differing strategies that BitSwap peers might employ have wildly different effects on the performance of the exchange as a whole. In BitTorrent, while a standard strategy is specified (tit-for-tat), a variety of others have been implemented, ranging from BitTyrant [8] (sharing the least-possible), to BitThief [8] (exploiting a vulnerability and never share), to PropShare [8] (sharing proportionally). A range of strategies (good and malicious) could similarly be implemented by BitSwap peers. The choice of function, then, should aim to:

1. maximize the trade performance for the node, and the whole exchange
2. prevent freeloaders from exploiting and degrading the exchange
3. be effective with and resistant to other, unknown strategies
4. be lenient to trusted peers

The exploration of the space of such strategies is future work. One choice of function that works in practice is a sigmoid, scaled by a *debt retio*:

Let the *debt ratio* r between a node and its peer be:

$$r = \frac{\texttt{bytes\_sent}}{\texttt{bytes\_recv} + 1}$$

Given r, let the probability of sending to a debtor be:

$$P\Big(\ send \mid r\ \Big) = 1 - \frac{1}{1 + exp(6 - 3r)}$$

As you can see in Figure 1, this function drops off quickly as the nodes' *debt ratio* surpasses twice the established credit.

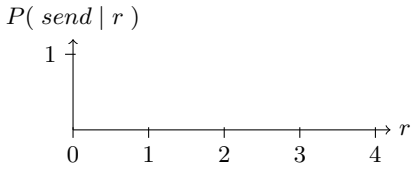


Figure 1: Probability of Sending as r increases

The *debt ratio* is a measure of trust: lenient to debts between nodes that have previously exchanged lots of data successfully, and merciless to unknown, untrusted nodes. This (a) provides resistance to attackers who would create lots of new nodes (sybill attacks), (b) protects previously successful trade relationships, even if one of the nodes is temporarily unable to provide value, and (c) eventually chokes relationships that have deteriorated until they improve.

3.4.3 BitSwap Ledger

BitSwap nodes keep ledgers accounting the transfers with other nodes. This allows nodes to keep track of history and avoid tampering. When activating a connection, BitSwap nodes exchange their ledger information. If it does not match exactly, the ledger is reinitialized from scratch, losing the accrued credit or debt. It is possible for malicious nodes to purposefully "lose" the Ledger, hoping to erase debts. It is unlikely that nodes will have accrued enough debt to warrant also losing the accrued trust; however the partner node is free to count it as misconduct, and refuse to trade.

```
type Ledger struct {
  owner      NodeId
  partner    NodeId
  bytes_sent int
  bytes_recv int
  timestamp  Timestamp
}
```

Nodes are free to keep the ledger history, though it is not necessary for correct operation. Only the current ledger entries are useful. Nodes are also free to garbage collect ledgers as necessary, starting with the less useful ledgers: the old (peers may not exist anymore) and small.

3.4.4 BitSwap Specification

BitSwap nodes follow a simple protocol.

```
// Additional state kept
type BitSwap struct {
  ledgers map[NodeId]Ledger
  // Ledgers known to this node, inc inactive

  active map[NodeId]Peer
  // currently open connections to other nodes

  need_list []Multihash
  // checksums of blocks this node needs

  have_list []Multihash
  // checksums of blocks this node has
}

type Peer struct {
  nodeid NodeId
  ledger Ledger
  // Ledger between the node and this peer

  last_seen Timestamp
  // timestamp of last received message

  want_list []Multihash
  // checksums of all blocks wanted by peer
  // includes blocks wanted by peer's peers
}

// Protocol interface:
interface Peer {
  open (nodeid :NodeId, ledger :Ledger);
  send_want_list (want_list :WantList);
  send_block (block :Block) -> (complete :Bool);
  close (final :Bool);
}
```

Sketch of the lifetime of a peer connection:

1. Open: peers send `ledgers` until they agree.
2. Sending: peers exchange `want_lists` and `blocks`.
3. Close: peers deactivate a connection.
4. Ignored: (special) a peer is ignored (for the duration of a timeout) if a node's strategy avoids sending

Peer.open(NodeId, Ledger).

When connecting, a node initializes a connection with a `Ledger`, either stored from a connection in the past or a new one zeroed out. Then, sends an Open message with the `Ledger` to the peer.

Upon receiving an `Open` message, a peer chooses whether to activate the connection. If – acording to the receiver's `Ledger` – the sender is not a trusted agent (transmission below zero, or large outstanding debt) the receiver may opt to ignore the request. This should be done probabilistically with an `ignore_cooldown` timeout, as to allow errors to be corrected and attackers to be thwarted.

If activating the connection, the receiver initializes a Peer object with the local version of the `Ledger` and sets the `last_seen` timestamp. Then, it compares the received `Ledger` with its own. If they match exactly, the connections have opened. If they do not match, the peer creates a new zeroed out `Ledger` and sends it.

Peer.send_want_list(WantList).

While the connection is open, nodes advertise their `want_list` to all connected peers. This is done (a) upon opening the connection, (b) after a randomized periodic timeout, (c) after a change in the `want_list` and (d) after receiving a new block.

Upon receiving a `want_list`, a node stores it. Then, it checks whether it has any of the wanted blocks. If so, it sends them according to the *BitSwap Strategy* above.

Peer.send_block(Block).

Sending a block is straightforward. The node simply transmits the block of data. Upon receiving all the data, the receiver computes the Multihash checksum to verify it matches the expected one, and returns confirmation.

Upon finalizing the correct transmission of a block, the receiver moves the block from `need_list` to `have_list`, and both the receiver and sender update their ledgers to reflect the additional bytes transmitted.

If a transmission verification fails, the sender is either malfunctioning or attacking the receiver. The receiver is free to refuse further trades. Note that BitSwap expects to operate on a reliable transmission channel, so transmission errors – which could lead to incorrect penalization of an honest sender – are expected to be caught before the data is given to BitSwap.

Peer.close(Bool).

The `final` parameter to `close` signals whether the intention to tear down the connection is the sender's or not. If false, the receiver may opt to re-open the connection immediatelty. This avoids premature closes.

A peer connection should be closed under two conditions:

- a `silence_wait` timeout has expired without receiving any messages from the peer (default BitSwap uses 30 seconds). The node issues `Peer.close(false)`.
- the node is exiting and BitSwap is being shut down. In this case, the node issues `Peer.close(true)`.

After a `close` message, both receiver and sender tear down the connection, clearing any state stored. The `Ledger` may be stored for the future, if it is useful to do so.

Notes.

- Non-`open` messages on an inactive connection should be ignored. In case of a `send_block` message, the receiver may check the block to see if it is needed and correct, and if so, use it. Regardless, all such out-of-order messages trigger a `close(false)` message from the receiver to force re-initialization of the connection.

3.5 Object Merkle DAG

The DHT and BitSwap allow IPFS to form a massive peer-to-peer system for storing and distributing blocks quickly and robustly. On top of these, IPFS builds a Merkle DAG, a directed acyclic graph where links between objects are cryptographic hashes of the targets embedded in the sources. This is a generalization of the Git data structure. Merkle DAGs provide IPFS many useful properties, including:

1. **Content Addressing:** all content is uniquely identified by its `multihash` checksum, **including links**.
2. **Tamper resistance:** all content is verified with its checksum. If data is tampered with or corrupted, IPFS detects it.
3. **Deduplication:** all objects that hold the exact same content are equal, and only stored once. This is particularly useful with index objects, such as git `trees` and `commits`, or common portions of data.

The IPFS Object format is:

```
type IPFSLink struct {
  Name string
  // name or alias of this link

  Hash Multihash
  // cryptographic hash of target

  Size int
  // total size of target
}

type IPFSObject struct {
  links []IPFSLink
  // array of links

  data []byte
  // opaque content data
}
```

The IPFS Merkle DAG is an extremely flexible way to store data. The only requirements are that object references be (a) content addressed, and (b) encoded in the format above. IPFS grants applications complete control over the data field; applications can use any custom data format they chose, which IPFS may not understand. The separate in-object link table allows IPFS to:

- List all object references in an object. For example:

```
> ipfs ls /XLZ1625Jjn7SubMDgEyeaynFuR84ginqvzb
XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x 189458 less
XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5 19441 script
XLF4hwVHsVuZ78FZK6fozf8Jj9WEURMbCX4 5286 template

<object multihash> <object size> <link name>
```

- Resolve string path lookups, such as `foo/bar/baz`. Given an object, IPFS resolves the first path component to a hash in the object's link table, fetches that second object, and repeats with the next component. Thus, string paths can walk the Merkle DAG no matter what the data formats are.
- Resolve all objects referenced recursively:

```
> ipfs refs --recursive \
  /XLZ1625Jjn7SubMDgEyeaynFuR84ginqvzb
XLLxhdgJcXzLbtsLRL1twCHA2NrURp4H38s
XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x
XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5
XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z
...
```

A raw data field and a common link structure are the necessary components for constructing arbitrary data structures on top of IPFS. While it is easy to see how the Git object model fits on top of this DAG, consider these other potential data structures: (a) key-value stores (b) traditional relational databases (c) Linked Data triple stores (d) linked document publishing systems (e) linked communications platforms (f) cryptocurrency blockchains. These can all be modeled on top of the IPFS Merkle DAG, which allows any of these systems to use IPFS as a transport protocol for more complex applications.

3.5.1 Paths

IPFS objects can be traversed with a string path API. Paths work as they do in traditional UNIX filesystems and the Web. The Merkle DAG links make traversing it easy. Note that full paths in IPFS are of the form:

```
# format
/ipfs/<hash-of-object>/<name-path-to-object>

# example
/ipfs/XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x/foo.txt
```

The `/ipfs` prefix allows mounting into existing systems at a standard mount point without conflict (mount point names are of course configurable). The second path component (first within IPFS) is the hash of an object. This is always the case, as there is no global root. A root object would have the impossible task of handling consistency of millions of objects in a distributed (and possibly disconnected) environment. Instead, we simulate the root with content addressing. All objects are always accessible via their hash. Note this means that given three objects in path `<foo>/bar/baz`, the last object is accessible by all:

```
/ipfs/<hash-of-foo>/bar/baz
/ipfs/<hash-of-bar>/baz
/ipfs/<hash-of-baz>
```

3.5.2 Local Objects

IPFS clients require some *local storage*, an external system on which to store and retrieve local raw data for the objects IPFS manages. The type of storage depends on the node's use case. In most cases, this is simply a portion of disk space (either managed by the native filesystem, by a key-value store such as leveldb [4], or directly by the IPFS client). In others, for example non-persistent caches, this storage is just a portion of RAM.

Ultimately, all blocks available in IPFS are in some node's *local storage*. When users request objects, they are found, downloaded, and stored locally, at least temporarily. This provides fast lookup for some configurable amount of time thereafter.

3.5.3 Object Pinning

Nodes who wish to ensure the survival of particular objects can do so by `pinning` the objects. This ensures the objects are kept in the node's *local storage*. Pinning can be done recursively, to pin down all linked descendent objects as well. All objects pointed to are then stored locally. This is particularly useful to persist files, including references. This also makes IPFS a Web where links are *permanent*, and Objects can ensure the survival of others they point to.

3.5.4 Publishing Objects

IPFS is globally distributed. It is designed to allow the files of millions of users to coexist together. The DHT, with content-hash addressing, allows publishing objects in a fair, secure, and entirely distributed way. Anyone can publish an object by simply adding its key to the DHT, adding themselves as a peer, and giving other users the object's path. Note that Objects are essentially immutable, just like in Git. New versions hash differently, and thus are new objects. Tracking versions is the job of additional versioning objects.

3.5.5 Object-level Cryptography

IPFS is equipped to handle object-level cryptographic operations. An encrypted or signed object is wrapped in a special frame that allows encryption or verification of the raw bytes.

```
type EncryptedObject struct {
  Object []bytes
  // raw object data encrypted

  Tag []bytes
  // optional tag for encryption groups
}

type SignedObject struct {
  Object []bytes
  // raw object data signed

  Signature []bytes
  // hmac signature

  PublicKey []multihash
  // multihash identifying key
}
```

Cryptographic operations change the object's hash, defining a different object. IPFS automatically verifies signatures, and can decrypt data with user-specified keychains. Links of encrypted objects are protected as well, making traversal impossible without a decryption key. It is possible to have a parent object encrypted under one key, and a child under another or not at all. This secures links to shared objects.

3.6 Files

IPFS also defines a set of objects for modeling a versioned filesystem on top of the Merkle DAG. This object model is similar to Git's:

1. `block`: a variable-size block of data.
2. `list`: a collection of blocks or other lists.
3. `tree`: a collection of blocks, lists, or other trees.
4. `commit`: a snapshot in the version history of a tree.

I hoped to use the Git object formats exactly, but had to depart to introduce certain features useful in a distributed filesystem, namely (a) fast size lookups (aggregate byte sizes have been added to objects), (b) large file deduplication (adding a `list` object), and (c) embedding of `commits` into `trees`. However, IPFS File objects are close enough to Git that conversion between the two is possible. Also, a set of Git objects can be introduced to convert without losing any information (unix file permissions, etc).

Notation: File object formats below use JSON. Note that this structure is actually binary encoded using protobufs, though ipfs includes import/export to JSON.

3.6.1 File Object: `blob`

The `blob` object contains an addressable unit of data, and represents a file. IPFS Blocks are like Git blobs or filesystem data blocks. They store the users' data. Note that IPFS files can be represented by both `lists` and `blobs`. Blobs have no links.

```
{
  "data": "some data here",
  // blobs have no links
}
```

3.6.2 *File Object:* `list`

The `list` object represents a large or deduplicated file made up of several IPFS `blobs` concatenated together. `lists` contain an ordered sequence of `blob` or `list` objects. In a sense, the IPFS `list` functions like a filesystem file with indirect blocks. Since `lists` can contain other `lists`, topologies including linked lists and balanced trees are possible. Directed graphs where the same node appears in multiple places allow in-file deduplication. Of course, cycles are not possible, as enforced by hash addressing.

```
{
  "data": ["blob", "list", "blob"],
    // lists have an array of object types as data
  "links": [
    { "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x",
      "size": 189458 },
    { "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5",
      "size": 19441 },
    { "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z",
      "size": 5286 }
    // lists have no names in links
  ]
}
```

3.6.3 *File Object:* `tree`

The `tree` object in IPFS is similar to Git's: it represents a directory, a map of names to hashes. The hashes reference `blobs`, `lists`, other `trees`, or `commits`. Note that traditional path naming is already implemented by the Merkle DAG.

```
{
  "data": ["blob", "list", "blob"],
    // trees have an array of object types as data
  "links": [
    { "hash": "XLYkgq61DYaQ8NhkcqyU7rLcnSa7dSHQ16x",
      "name": "less", "size": 189458 },
    { "hash": "XLHBNmRQ5sJJrdMPuu48pzeyTtRo39tNDR5",
      "name": "script", "size": 19441 },
    { "hash": "XLWVQDqxo9Km9zLyquoC9gAP8CL1gWnHZ7z",
      "name": "template", "size": 5286 }
    // trees do have names
  ]
}
```

3.6.4 *File Object:* `commit`

The `commit` object in IPFS represents a snapshot in the version history of any object. It is similar to Git's, but can reference any type of object. It also links to author objects.

```
{
  "data": {
    "type": "tree",
    "date": "2014-09-20 12:44:06Z",
    "message": "This is a commit message."
  },
  "links": [
```

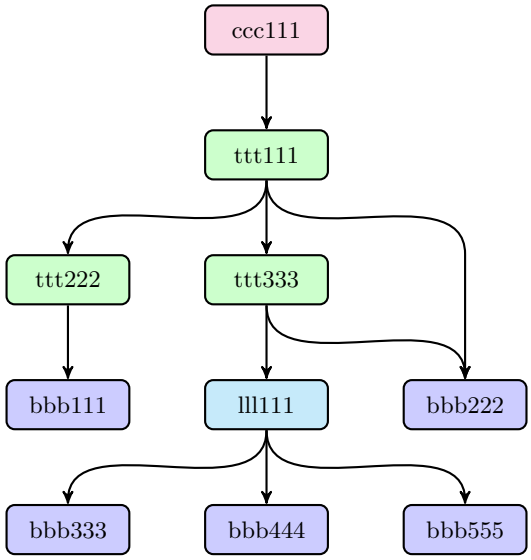


Figure 2: Sample Object Graph

```
> ipfs file-cat <ccc111-hash> --json
{
  "data": {
    "type": "tree",
    "date": "2014-09-20 12:44:06Z",
    "message": "This is a commit message."
  },
  "links": [
    { "hash": "<ccc000-hash>",
      "name": "parent", "size": 25309 },
    { "hash": "<ttt111-hash>",
      "name": "object", "size": 5198 },
    { "hash": "<aaa111-hash>",
      "name": "author", "size": 109 }
  ]
}

> ipfs file-cat <ttt111-hash> --json
{
  "data": ["tree", "tree", "blob"],
  "links": [
    { "hash": "<ttt222-hash>",
      "name": "ttt222-name", "size": 1234 },
    { "hash": "<ttt333-hash>",
      "name": "ttt333-name", "size": 3456 },
    { "hash": "<bbb222-hash>",
      "name": "bbb222-name", "size": 22 }
  ]
}

> ipfs file-cat <bbb222-hash> --json
{
  "data": "blob222 data",
  "links": []
}
```

Figure 3: Sample Objects

```
    { "hash": "XLa1qMBKiSEEDhojb9FFZ4tEvLf7FEQdhdU",
      "name": "parent", "size": 25309 },
    { "hash": "XLGw74KAy9junbh28x7ccWov9inu1Vo7pnX",
      "name": "object", "size": 5198 },
    { "hash": "XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm",
      "name": "author", "size": 109 }
  ]
}
```

3.6.5 Version control

The `commit` object represents a particular snapshot in the version history of an object. Comparing the objects (and children) of two different commits reveals the differences between two versions of the filesystem. As long as a single `commit` and all the children objects it references are accessible, all preceding versions are retrievable and the full history of the filesystem changes can be accessed. This falls out of the Merkle DAG object model.

The full power of the Git version control tools is available to IPFS users. The object model is compatible, though not the same. It is possible to (a) build a version of the Git tools modified to use the IPFS object graph, (b) build a mounted FUSE filesystem that mounts an IPFS `tree` as a Git repo, translating Git filesystem read/writes to the IPFS formats.

3.6.6 Filesystem Paths

As we saw in the Merkle DAG section, IPFS objects can be traversed with a string path API. The IPFS File Objects are designed to make mounting IPFS onto a UNIX filesystem simpler. They restrict `trees` to have no data, in order to represent them as directories. And `commits` can either be represented as directories or hidden from the filesystem entirely.

3.6.7 Splitting Files into Lists and Blob

One of the main challenges with versioning and distributing large files is finding the right way to split them into independent blocks. Rather than assume it can make the right decision for every type of file, IPFS offers the following alternatives:

(a) Use Rabin Fingerprints [**?**] as in LBFS [**?**] to pick suitable block boundaries.

(b) Use the rsync [**?**] rolling-checksum algorithm, to detect blocks that have changed between versions.

(c) Allow users to specify block-splitting functions highly tuned for specific files.

3.6.8 Path Lookup Performance

Path-based access traverses the object graph. Retrieving each object requires looking up its key in the DHT, connecting to peers, and retrieving its blocks. This is considerable overhead, particularly when looking up paths with many components. This is mitigated by:

- **tree caching**: since all objects are hash-addressed, they can be cached indefinitely. Additionally, `trees` tend to be small in size so IPFS prioritizes caching them over `blobs`.
- **flattened trees**: for any given `tree`, a special `flattened tree` can be constructed to list all objects reachable from the `tree`. Names in the `flattened tree` would really be paths parting from the original tree, with slashes.

For example, `flattened tree` for `ttt111` above:

```
{
"data":
  ["tree", "blob", "tree", "list", "blob" "blob"],
"links": [
  { "hash": "<ttt222-hash>", "size": 1234
    "name": "ttt222-name" },
  { "hash": "<bbb111-hash>", "size": 123,
    "name": "ttt222-name/bbb111-name" },
  { "hash": "<ttt333-hash>", "size": 3456,
    "name": "ttt333-name" },
  { "hash": "<lll111-hash>", "size": 587,
    "name": "ttt333-name/lll111-name"},
  { "hash": "<bbb222-hash>", "size": 22,
    "name": "ttt333-name/lll111-name/bbb222-name" },
  { "hash": "<bbb222-hash>", "size": 22
    "name": "bbb222-name" }
] }
```

3.7 IPNS: Naming and Mutable State

So far, the IPFS stack forms a peer-to-peer block exchange constructing a content-addressed DAG of objects. It serves to publish and retrieve immutable objects. It can even track the version history of these objects. However, there is a critical component missing: mutable naming. Without it, all communication of new content must happen off-band, sending IPFS links. What is required is some way to retrieve mutable state at *the same path*.

It is worth stating why – if mutable data is necessary in the end – we worked hard to build up an *immutable* Merkle DAG. Consider the properties of IPFS that fall out of the Merkle DAG: objects can be (a) retrieved via their hash, (b) integrity checked, (c) linked to others, and (d) cached indefinitely. In a sense:

Objects are **permanent**

These are the critical properties of a high-performance distributed system, where data is expensive to move across network links. Object content addressing constructs a web with (a) significant bandwidth optimizations, (b) untrusted content serving, (c) permanent links, and (d) the ability to make full permanent backups of any object and its references.

The Merkle DAG, immutable content-addressed objects, and Naming, mutable pointers to the Merkle DAG, instantiate a dichotomy present in many successful distributed systems. These include the Git Version Control System, with its immutable objects and mutable references; and Plan9 [**?**], the distributed successor to UNIX, with its mutable Fossil [**?**] and immutable Venti [**?**] filesystems. LBFS [**?**] also uses mutable indices and immutable chunks.

3.7.1 Self-Certified Names

Using the naming scheme from SFS [12, 11] gives us a way to construct self-certified names, in a cryptographically assigned global namespace, that are mutable. The IPFS scheme is as follows.

1. Recall that in IPFS:

```
NodeId = hash(node.PubKey)
```

2. We assign every user a mutable namespace at:

```
/ipns/<NodeId>
```

3. A user can publish an Object to this path **Signed** by her private key, say at:

```
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/
```

4. When other users retrieve the object, they can check the signature matches the public key and NodeId. This verifies the authenticity of the Object published by the user, achieving mutable state retrival.

Note the following details:

- The `ipns` (InterPlanetary Name Space) separate prefix is to establish an easily recognizable distinction between *mutable* and *immutable* paths, for both programs and human readers.
- Because this is *not* a content-addressed object, publishing it relies on the only mutable state distribution system in IPFS, the Routing system. The process is (1) publish the object as a regular immutable IPFS object, (2) publish its hash on the Routing system as a metadata value:

```
routing.setValue(NodeId, <ns-object-hash>)
```

- Any links in the Object published act as sub-names in the namespace:

```
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/docs
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/docs/ipfs
```

- it is advised to publish a `commit` object, or some other object with a version history, so that clients may be able to find old names. This is left as a user option, as it is not always desired.

Note that when users publish this Object, it cannot be published in the same way

3.7.2 Human Friendly Names

While IPNS is indeed a way of assigning and reassigning names, it is not very user friendly, as it exposes long hash values as names, which are notoriously hard to remember. These work for URLs, but not for many kinds of offline transmission. Thus, IPFS increases the user-friendliness of IPNS with the following techniques.

Peer Links.

As encouraged by SFS, users can link other users' Objects directly into their own Objects (namespace, home, etc). This has the benefit of also creating a web of trust (and supports the old Certificate Authority model):

```
# Alice links to bob Bob
ipfs link /<alice-pk-hash>/friends/bob /<bob-pk-hash>

# Eve links to Alice
ipfs link /<eve-pk-hash/friends/alice /<alice-pk-hash>

# Eve also has access to Bob
/<eve-pk-hash/friends/alice/friends/bob

# access Verisign certified domains
/<verisign-pk-hash>/foo.com
```

DNS TXT IPNS Records.

If `/ipns/<domain>` is a valid domain name, IPFS looks up key `ipns` in its `DNS TXT` records. IPFS interprets the value as either an object hash or another IPNS path:

```
# this DNS TXT record
ipfs.benet.ai. TXT "ipfs=XLF2ipQ4jD3U ..."

# behaves as symlink
ln -s /ipns/XLF2ipQ4jD3U /ipns/fs.benet.ai
```

Proquint Pronounceable Identifiers.

There have always been schemes to encode binary into pronounceable words. IPNS supports Proquint [**?**]. Thus:

```
# this proquint phrase
/ipns/dahih-dolij-sozuk-vosah-luvar-fuluh

# will resolve to corresponding
/ipns/KhAwNprxYVxKqpDZ
```

Name Shortening Services.

Services are bound to spring up that will provide name shortening as a service, offering up their namespaces to users. This is similar to what we see today with DNS and Web URLs:

```
# User can get a link from
/ipns/shorten.er/foobar

# To her own namespace
/ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm
```

3.8 Using IPFS

IPFS is designed to be used in a number of different ways. Here are just some of the usecases I will be pursuing:

1. As a mounted global filesystem, under `/ipfs` and `/ipns`.
2. As a mounted personal sync folder that automatically versions, publishes, and backs up any writes.
3. As an encrypted file or data sharing system.
4. As a versioned package manager for *all* software.
5. As the root filesystem of a Virtual Machine.
6. As the boot filesystem of a VM (under a hypervisor).

7. As a database: applications can write directly to the Merkle DAG data model and get all the versioning, caching, and distribution IPFS provides.

8. As a linked (and encrypted) communications platform.

9. As an integrity checked CDN for large files (without SSL).

10. As an encrypted CDN.

11. On webpages, as a web CDN.

12. As a new Permanent Web where links do not die.

The IPFS implementations target:

(a) an IPFS library to import in your own applications.

(b) commandline tools to manipulate objects directly.

(c) mounted file systems, using FUSE [**?**] or as kernel modules.

4. THE FUTURE

The ideas behind IPFS are the product of decades of successful distributed systems research in academia and open source. IPFS synthesizes many of the best ideas from the most successful systems to date. Aside from BitSwap, which is a novel protocol, the main contribution of IPFS is this coupling of systems and synthesis of designs.

IPFS is an ambitious vision of new decentralized Internet infrastructure, upon which many different kinds of applications can be built. At the bare minimum, it can be used as a global, mounted, versioned filesystem and namespace, or as the next generation file sharing system. At its best, it could push the web to new horizons, where publishing valuable information does not impose hosting it on the publisher but upon those interested, where users can trust the content they receive without trusting the peers they receive it from, and where old but important files do not go missing. IPFS looks forward to bringing us toward the Permanent Web.

5. ACKNOWLEDGMENTS

IPFS is the synthesis of many great ideas and systems. It would be impossible to dare such ambitious goals without standing on the shoulders of such giants. Personal thanks to David Dalrymple, Joe Zimmerman, and Ali Yahya for long discussions on many of these ideas, in particular: exposing the general Merkle DAG (David, Joe), rolling hash blocking (David), and s/kademlia sybill protection (David, Ali). And special thanks to David Mazieres, for his ever brilliant ideas.

6. REFERENCES TODO

7. REFERENCES

[1] I. Baumgart and S. Mies. S/kademlia: A practicable approach towards secure key-based routing. In *Parallel and Distributed Systems, 2007 International Conference on*, volume 2, pages 1–8. IEEE, 2007.

[2] I. BitTorrent. Bittorrent and Âţtorrent software surpass 150 million user milestone, Jan. 2012.

[3] B. Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[4] J. Dean and S. Ghemawat. leveldb–a fast and lightweight key/value database library by google, 2011.

[5] M. J. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with coral. In *NSDI*, volume 4, pages 18–18, 2004.

[6] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, et al. Oceanstore: An architecture for global-scale persistent storage. *ACM Sigplan Notices*, 35(11):190–201, 2000.

[8] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. Bittorrent is an auction: analyzing and improving bittorrent's incentives. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 243–254. ACM, 2008.

[9] A. J. Mashtizadeh, A. Bittau, Y. F. Huang, and D. Mazieres. Replication, history, and grafting in the ori file system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 151–166. ACM, 2013.

[10] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[11] D. Mazieres and F. Kaashoek. Self-certifying file system. 2000.

[12] D. Mazieres and M. F. Kaashoek. Escaping the evils of centralized control with self-certifying pathnames. In *Proceedings of the 8th ACM SIGOPS European workshop on Support for composing distributed applications*, pages 118–125. ACM, 1998.

[13] J. Rosenberg and A. Keranen. Interactive connectivity establishment (ice): A protocol for network address translator (nat) traversal for offer/answer protocols. 2013.

[14] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low extra delay background transport (ledbat). *draft-ietf-ledbat-congestion-04. txt*, 2010.

[15] R. R. Stewart and Q. Xie. *Stream control transmission protocol (SCTP): a reference guide*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[16] L. Wang and J. Kangasharju. Measuring large-scale distributed systems: case of bittorrent mainline dht. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

Blockchain Consensus Algorithms: A Survey

Md Sadek Ferdous, *Member, IEEE,* Mohammad Jabed Morshed Chowdhury,
Mohammad A. Hoque, *Member, IEEE*, and Alan Colman

Abstract—In recent years, blockchain technology has received unparalleled attention from academia, industry, and governments all around the world. It is considered a technological breakthrough anticipated to disrupt several application domains touching all spheres of our lives. The sky-rocket anticipation of its potential has caused a wide-scale exploration of its usage in different application domains. This has resulted in a plethora of blockchain systems for various purposes. However, many of these blockchain systems suffer from serious shortcomings related to their performance and security, which need to be addressed before any wide-scale adoption can be achieved. A crucial component of any blockchain system is its underlying consensus algorithm, which in many ways, determines its performance and security. Therefore, to address the limitations of different blockchain systems, several existing as well novel consensus algorithms have been introduced. A systematic analysis of these algorithms will help to understand how and why any particular blockchain performs the way it functions. However, the existing studies of consensus algorithms are not comprehensive. Those studies have incomplete discussions on the properties of the algorithms and fail to analyse several major blockchain consensus algorithms in terms of their scopes. This article fills this gap by analysing a wide range of consensus algorithms using a comprehensive taxonomy of properties and by examining the implications of different issues still prevalent in consensus algorithms in detail. The result of the analysis is presented in tabular formats, which provides a visual illustration of these algorithms in a meaningful way. We have also analysed more than hundred top crypto-currencies belonging to different categories of consensus algorithms to understand their properties and to implicate different trends in these crypto-currencies. Finally, we have presented a decision tree of algorithms to be used as a tool to test the suitability of consensus algorithms under different criteria.

Index Terms—Blockchain, Distributed Consensus, Proof of Work, PoW, Proof of Stake, PoS, Delegated Proof of Stake, DPoS.

1 Introduction

In the last few years, blockchain has received wide-spread attention among the industry, the Government, and academia alike. This interest has been piqued by the success of Bitcoin [1] that was introduced in 2008. While crypto-currencies have emerged as the principal and the most popular application of blockchain technology, many enthusiasts from different disciplines have identified and proposed a plethora of applications of blockchain in a multitude of application domains [2], [3]. The possibility of exploiting blockchain in so many areas has created huge anticipation surrounding blockchain systems. Indeed, it is regarded as one of the fundamental technologies to revolutionise the landscapes of the identified application domains.

A blockchain system is, fundamentally, a distributed system that relies on a consensus algorithm that ensures agreement on the states of certain data among distributed nodes. A consensus algorithm is the core component that directly dictates how such a system behaves and the performance it can achieve. Distributed consensus has been a widely studied research topic in distributed systems, however, with the advent of blockchain, it has received renewed attention. A wide variety of crypto-currencies targeting different application domains has introduced an array of unique requirements that can only be satisfied by their corresponding consensus mechanisms. This fact has fuelled the need not only to examine the applicability of existing consensus algorithms in newer settings, but also to innovate novel consensus algorithms. Consequently, several consensus algorithms have emerged, each of which possesses interesting properties and unique capabilities.

As the characteristics of various types of blockchain systems are fundamentally dependent on the consensus algorithms they use, a systematic analysis of existing consensus algorithms is required. It is necessary to examine, compare, and contrast these algorithms. There have a been a number of attempts aiming to fulfil this goal can be found in [4], [5], [6], [7], [8], [9], [10]. In particular, the works carried out by Cachin et al. [4] and Bano et al. [5] are noteworthy as they represent the pioneer works in this scope. Cachin et al., in their work, have explored different aspects of distributed systems and consensus and focused on consensus algorithm deployed in blockchain systems that are not to open to the public. On the other hand, the focus of the work by Bano et al. is more general in the sense they have explored consensus algorithms used both in public as well as private systems. Another exceptional work is by Wang et al. [6] in which the authors have presented a comprehensive survey of different aspects of consensus, mining, and blockchains in a detailed fashion.

However, all these works have some major shortcom-

M. S. Ferdous is with Shahjalal University of Science and Technology, Sylhet 3114, Bangladesh and Imperial College London, London SW7 2AZ, U.K. E-mail: sadek-cse@sust.edu.

Mohammad Jabed Morshed Chowdhury is with La Trobe University, Melbourne, Victoria-3086, Australia. E-mail: m.chowdhury@latrobe.edu.au.

Mohammad A. Hoque is with University of Helsinki, 3835 Helsinki, Helsinki Finland. E-mail: mohammad.a.hoque@helsinki.fi.

Alan Coleman is with Swinburne University of Technology, Hawthorn, Australia. E-mail: acolman@swin.edu.au.

ings. For example, the factors upon which the consensus algorithms have been analysed are not comprehensive. Importantly, a wide range of consensus algorithms and their internal mechanisms utilised in many existing crypto-currencies have not been considered at all. In addition, all of these studies have failed to capture the practical interrelation between blockchain systems (mostly crypto-currencies) and their corresponding consensus algorithms. All in all, there is a pressing need for a study that analyses a wide range of existing consensus algorithms and the blockchain systems in a practical-oriented way and synthesises this analyses into a conceptual framework in a concise yet comprehensive manner. The principal motivation of this article is to fill in this gap.

Contributions. The main contributions of the article are presented below:

- A novel taxonomy of consensus properties, capturing different aspects of a consensus algorithm, has been created. In this taxonomy, consensus algorithms have been categorised in two major categories: incentivised and non-incentivised algorithms, which have been again sub-divided as per different considerations. Consensus algorithms belonging to each sub-category analysed together using the taxonomy of consensus properties.
- The analysis of each sub-category has been summarised in tabular formats so as to visually represent it in a comprehensible way.
- For each category (and the sub-category, if any), the corresponding blockchain systems (predominantly crypto-currencies) have been analysed as well. The analysis result has been presented in a concise fashion, which can be used to understand the inter-relation between these systems and their underlying consensus algorithms.
- The major issues in each category of consensus algorithm have been examined in detail, and their implications have been further analysed.
- Over hundred crypto-currencies, belonging to different consensus algorithms, have been examined to understand their different properties. These properties then have been utilised to analyse and identify different trends among these crypto-currencies.
- Finally, a decision tree of consensus algorithms have been presented. This tree can be utilised to test the suitability of a consensus algorithm under certain criteria.

In short, with these contributions, this article represents one of the most comprehensive studies of blockchain consensus algorithms as of now.

Structure. In Section 2 we present a brief background on distributed consensus, highlighting its different components, types and properties. Section 3 outlines a brief presentation on blockchain covering its different aspects such as types, properties, layers. A taxonomy of consensus algorithms and their underlying properties is presented in Section 4. Section 5 and Section 6 analyse different incentivised consensus algorithm whereas Section 7 examines the different non-incentivised consensus algorithms. Finally, we conclude in Section 8 with a detailed discussion on different issues involving the analysed consensus algorithms and the corresponding crypto-currencies.

2 Background: Distributed Consensus

Consensus mechanisms in distributed systems have been a well studied research problem for nearly three decades. Such mechanisms enable consensus to be achieved regarding a shared state/data among a set of distributed nodes. The need for a shared state originated the notion of replicated database systems in order to ensure resilience against node failures within a network. Such database systems ensure that data is not lost when one or more nodes fail to function in an excepted fashion.

The notion of the replicated database can be generalised with the concept of State Machine Replication (SMR) [11]. The core idea behind SMR is that a computing machine can be expressed as a deterministic *state machine*. The machine accepts an input message, performs its predefined computation, and might produce an output/response. These actions essentially change its state. SMR conceptualises that such a state machine, with an initial state, can be replicated among different nodes. If it can be ensured that all the participating nodes receive the same set of input messages in the exact same order (the phenomenon known as *atomic broadcast*), then each node would be able to evolve the states of its state machine individually in exactly the same fashion. This can guarantee consistency and availability regarding the state of the machine (as well as data it holds) among all (applicable) nodes even in the presence of node failures. Once this occurs, it can be said that a distributed consensus has emerged among the participating nodes. It is imperative that a protocol is defined to ensure timely dissemination and atomic broadcast of input messages among the nodes and, in many ways, dictates how a distributed consensus is achieved and maintained. Hence, such a protocol is aptly called a consensus protocol.

Designing and deploying a consensus protocol is a challenging task as it needs to consider several crucial issues such as resiliency against node failures, node behaviour, network partitioning, network latency, corrupt or out-of-order inputs, and so on [7]. Schneider pointed out that there are two crucial requirements to reach and maintain consensus among distributed nodes. The first requirement is a deterministic state machine. The second requirement is a *consensus protocol* to disseminate inputs in a timely fashion and to ensure atomic broadcast among the participating nodes. At the same time, the consensus protocols must ensure the properties of the atomic broadcast [12], [13], [4], [5]. The properties of atomic broadcast in distributed consensus is illustrated in Table 1.

One way to achieve the design goals of such a protocol is to make certain assumptions under which the protocol is proved to function properly. These assumptions influence the critical characteristics of a consensus protocol. Next, we explore two sets of widely-used assumptions for any distributed consensus protocol.

The first set of assumptions are about the underlying networking type. Dwork et al. categorised three types of networks exhibiting different properties: synchronous, asynchronous, and partially/eventually synchronous [22]. The latency involved in delivering a message to all nodes

Properties	Note
Validity	This guarantees that if a message is broadcast by a valid node, it will be correctly included within the consensus protocol.
Agreement	This is to guarantee that if a message is delivered to a valid node, it will ultimately be delivered to all valid nodes.
Integrity	This is to ensure that a message is broadcast only once by a valid node.
Total Order	This is to ensure that all nodes agree to the order of all delivered messages.

Table 1: Atomic broadcast Properties of Distributed Consensus Protocols.

Properties	Note
Safety/ Consistency	A consensus protocol is considered safe (or consistent) only when all nodes produce the same valid output, according to the protocol rules, for the same atomic broadcast.
Liveness/ availability	If all non-faulty participating nodes produce an output (indicating the termination of the protocol), the protocol is considered live.
Fault Tolerance	It exhibits the network's capability to perform as intended in the midst of node failures.

Table 2: Properties of Distributed Consensus Protocols.

in a synchronous network is bound by some time denoted as Δ. On the other hand, the latency in an asynchronous network cannot be reliably bound by anyΔ. Finally, in a partially/eventually synchronous network, it is assumed that the network will eventually act as a synchronous network, even though it might be asynchronous over some arbitrary period of time.

The second set of assumptions is about the different properties of a consensus protocol. According to [7], a consensus protocol should have the following three properties; namely consistency, availability, and fault tolerance. These properties are elaborated in Table 2 A well-known theorem, by Fischer, Lynch and Paterson [23], called *FLP Impossibility* has shown that a deterministic consensus protocol cannot satisfy all three properties described above in an asynchronous network. It is more common to tend to favour safety and liveness over fault tolerance in the domain of distributed system applications. A related theorem is the CAP theorem [24], which states that a shared replicated datastore (or, more generally, a replicated state machine) cannot achieve both consistency and availability when a network partitions in such a way that an arbitrary number of messages might be dropped.

In addition to the above assumptions, there are two major fault-tolerance models within distributed systems: crash failure (or tolerance) and Byzantine failure [5], [7], [4]. The crash failure model deals with nodes that simply fail to respond due to some hardware or software failures. It may happen any time without any prior warning, and the corresponding node remains unresponsive until further actions are taken. Byzantine failure, on the other hand, deals with nodes that misbehave due to some software bugs or because of the nodes being compromised by an adversary. This type of failure was first identified and formalised by Leslie Lamport in his seminal paper with a metaphorical Byzantine General's problems [14]. A Byzantine node can behave maliciously by arbitrarily sending deceptive messages to others, which might affect the security of distributed systems. Hence, such nodes are mostly relevant in application with security implications.

To handle these two failure models, two corresponding major types of consensus mechanisms have emerged: Crash-tolerant consensus and Byzantine consensus [4]. Next, we briefly discuss each of them, along with their associated properties.

1) **Crash-tolerant consensus:** Algorithms belonging to this class aim to guarantee the atomic broadcast (total order) of messages within the participating nodes in the presence of certain number of node failures. These algorithms utilise the notion of views or epochs, which imply a certain duration of time or events. A leader is selected for each epoch who takes decisions regarding the atomic broadcast, and all other nodes comply with its decision. In case a leader fails due to a crash failure, the protocols elect a new leader to function. The best known algorithms belonging to this class can continue to function if the following condition holds: $t < n/2$ where t is the number of faulty nodes and n is the total number of participating nodes [4]. Examples of some well-known crash-tolerant consensus protocol are: Paxos [15], [16], Viewstamped Replication [17], ZooKeeper [18], and Raft [19].
2) **Byzantine consensus:** This class of algorithms aim to reach consensus amid of certain nodes exhibiting Byzantine behaviour. Such Byzantine nodes are assumed to be under the control of an adversary and behave unpredictably with malicious intent. Similar to any crash-tolerant consensus protocol, these protocols also utilise the concept of views/epochs where a leader is elected in each view to order messages for atomic broadcast, and other honest nodes are assumed to follow the instructions from the leader. One of the most well-known algorithms under this class is called Practical Byzantine Fault Tolerant (PBFT), which can achieve consensus in the presence of a certain number of Byzantine nodes under an eventual synchronous network assumption [20]. The tolerance level of PBFT is $f < n/3$, where f the number of Byzantine nodes and n denotes the number of total nodes participating in the network [4]. As we will explore later, PBFT algorithms have been widely utilised in different blockchain systems.

3 Background: Blockchain

In this section, we present a brief introduction to the blockchain technology and it related terminologies. At the centre of the blockchain technology is the blockchain itself stored by the nodes of a P2P network. A blockchain is a type of distributed ledger consisting of consecutive blocks chained together following a strict set of rules. Here, each block is created at a predefined interval, or after an event occurs, in a decentralised fashion by means of a consensus algorithm. Within each block, there are transactions by which a value is transferred in case of crypto-currencies or a data is stored for other blockchain systems. The consensus algorithm guar-

antees several data integrity related properties (discussed below) in blockchain.

Even though the terms blockchain and DLT (Distributed Ledger Technology) are used inter-changeably in the literature, there is a subtle difference between them which is worth highlighting. A blockchain is just an example of a particular type of ledger, there are other types of ledger. When a ledger (including a blockchain) is distributed across a network, it can be regarded as a Distributed Ledger.

Since the blockchain technology has been introduced with Bitcoin, it will be useful to understand how Bitcoin works. In Section 3.1, we discuss a brief primer of Bitcoin and its associated terminologies. Then, we describe different properties and types of blockchains in Section 3.2 and Section 3.3 respectively. Finally, we present the concept of blockchain layers in Section 3.4.

3.1 Bitcoin

The Bitcoin network consists of nodes within a P2P (Peer-to-Peer) network. Each node needs to download the Bitcoin software to connect to the network. There are different types of nodes in the network, with miner nodes and general nodes being the major ones. A general node is mostly used by users to transfer bitcoin in the network, whereas a miner node is a special node used for mining bitcoins (see below).

Each user within a node needs to utilise wallet software to create identities. An identity in the Bitcoin network consists of a private/public key pair, and a bitcoin address is derived from the corresponding public key. A sender needs to know such an address of the receiver to transfer any bitcoin. Bitcoin is transferred between two entities using the notion of a transaction where the sender utilises a wallet software for this purpose. This transaction is propagated to the network, which is collected by all miner nodes. Each miner node combines these transactions into a block and then engages in solving a cryptographic puzzle, with other miners, in which it tries to generate a random number which satisfies the required condition (the random number must be less than a target value called the *difficult target*). When a miner successfully solves the puzzle, that miner is said to have generated a valid block which is then propagated in the network. The Bitcoin protocol generates a certain amount of new Bitcoins for each new valid block and rewards the miner for its effort in creating the block. Other miners validate this newly mined block and then add it to the blockchain. Each new block refers to the last block in the chain, which in turn refers to its previous block, and so on. The very first block in the chain, known as the *genesis block*, however, has no such reference.

The decentralised nature of this mining process might result in multiple valid blocks generated by different miners and propagated at the same time in the network. All of them are added to the blockchain and they refer to the same last block in the chain. Consequently, multiple branches emerge from the same blockchain. This is a natural phenomenon in blockchain and is aptly known as *fork*. The fundamental goal of the corresponding consensus protocol is to resolve this fork so that only one branch remains and other branches are discarded. The consensus algorithm utilised in Bitcoin follows a simple rule: it lets the branches grow. As soon as one branch grows longer than the others (more specifically, the total cumulative computational effort of one branch exceeds the others), all miners select the longest branch (or the branch with the highest computational effort), discarding all other branches. Such a branch is known as the main branch and other branches are known as orphan branches. Only the miners in the main branch are entitled to receive their Bitcoin rewards. When a fork is resolved across the network, a distributed consensus emerges in the network.

The frequency of Bitcoin block generation depends on the difficulty parameter, which is adjusted after 2016 blocks. The protocol adjusts the difficulty parameter in such a way that a block is generated in every 10 minutes on average. However, the Bitcoin reward is halved after every $210,000$ blocks, or approximately after every 4 years. At the initial stage, the reward for generating a valid block had been 50 Bitcoins, which was halved to 25 Bitcoins in 2012 and 12.5 Bitcoins in 2016. The next halving will occur in 2020 where the reward will be reduced to 6.25 bitcoins per block. This geometric reduction in every four years underlines a maximum total supply of 21 million of Bitcoins. It is expected that this supply will be exhausted in the year of 2140 when the rewarded bitcoin will be infinitesimally small for each block.

The process of Bitcoin protocol is presented in Figure 1.

Figure 1: Block Generation Process of Bitcoin

A Each miner collects transactions that are broadcast in the network and uses her hashpower to try to generate a block via repeated invocation of a hash function on data. The data consists of the transactions that she saw fit to include, the hash of the previous block, her public key address, and a nonce.

B When a miner succeeds in generating a block, meaning that the hash of her block data is smaller than the current difficulty target, she broadcasts her block to the network.

C The other miners continue to extend the blockchain from this new block, only if they find that this block is valid, i.e., it refers the hash of the previous block of the longest chain and meets the current difficulty target.

D The block reward (newly minted coins) and the fees from the transactions go to the miner's public address. This means that only the miner can spend those by signing with her corresponding private key.

E The difficulty level readjusts according to the mining power of the participates, by updating the hash target value every 2016 blocks (≈2 weeks) so that blocks get generated once every 10 minutes on average.

F The block reward starts at 50 coins and halves in every 210, 000 blocks, i.e., about every 4 years.

3.2 Properties of blockchain

A blockchain exhibits several properties that make it a suitable candidate for several application domains [25]. The properties are discussed below.

- **Distributed consensus on the chain state**: One of the crucial properties of any blockchain is its capability to achieve a distributed consensus on the state of the chain without being reliant on any trusted third party. This opens up the door of opportunities to build and utilise a system where states and interactions are verifiable by the miners in public blockchain systems or by the authorised entities in private blockchain systems.
- **Immutability and irreversibility of chain state**: Achieving a distributed consensus with the participation of a large number of nodes ensures that the chain state becomes practically immutable and irreversible after a certain period of time. This also applies to smart-contracts and hence enabling the deployment and execution of immutable computer programs.
- **Data (transaction) persistence**: Data in a blockchain is stored in a distributed fashion, ensuring data persistence as long as there are participating nodes in the P2P network.
- **Data provenance**: The data storage process in any blockchain is facilitated by means of a mechanism called the transaction. Every transaction needs to be digitally signed using public key cryptography, which ensures the authenticity of the source of data. Combining this with the immutability and irreversibility of a blockchain provides a strong non-repudiation instrument for any data in the blockchain.
- **Distributed data control**: A blockchain ensures that data stored in the chain or retrieved from the chain can be carried out in a distributed manner that exhibits no single point of failure.
- **Accountability and transparency**: Since the state of the chain, along with every single interactions among participating entities, can be verified by any authorised entity, a blockchain promotes accountability and transparency.

3.3 Blockchain type

Depending on the application domains, different blockchain deployment strategies can be pursued. Based on these strategies, there are predominantly two types of blockchains, namely Public and Private blockchain, as discussed below:

- **Public blockchain**: A public blockchain, also known as the *Unpermissioned or permissionless Blockchain*, allows anyone to participate in the blockchain to create and validate blocks as well as to modify the chain state by storing and updating data through transactions among participating entities. This means that the blockchain state and its transactions, along with the data stored is transparent and accessible to everyone. This raises privacy concerns for particular scenarios where the privacy of such data needs to be preserved.
- **Private blockchain**: A private blockchain, also known as the *Permissioned Blockchain*, has a restrictive notion in comparison to its public counterpart in the sense that only authorised and trusted entities can participate in the activities within the blockchain. By allowing only authorised entities to participate in activities within the blockchain, a private blockchain can ensure the privacy of chain data, which might be desirable in some use-cases.

3.4 Blockchain Layers

There are several components in a blockchain system whose functionalities range from collecting transactions, propagating blocks, mining, achieving consensus and maintaining the ledger for its underlying crypto-currencies, and so on. These components can be grouped together according to their functionalities using different layers similar to the well-known TCP/IP layer. In fact, there have been a few suggestions to design a blockchain system using a layered approach [26], [27]. The motivation is that a layered design will be much more modular and easier to maintain. For example, in case a bug is found in a component of a layer in a blockchain system, it will only affect the functionalities of that corresponding layer while other layers remain unaffected. For example, David et al. [27] suggest four layers: consensus, mining, propagation, and semantic. However, we believe that the proposed layers do not reflect the proper grouping of functionalities. For example, consensus and mining should be part of the same layer as mining can be considered an inherent part of achieving consensus. In addition to this, some blockchain systems might not have any mining algorithms associated with it. In this paper, we, therefore, will define four layers (Figure 2): network, consensus, application, and meta-application. The functionalities of these layers are briefly presented below.

Meta-Application Layer: The functionalities of the meta-application layer in a blockchain system (see Figure 2) is to provide an overlay on top of the application layer to exploit the semantic interpretation of a blockchain system for other purposes in other application domains. For example, Bitcoin has been experimented to adopt in multiple application domains, such as DNS like decentralised naming system (Namecoin [28]), decentralised immutable time-stamped hashed record (Proof of Existence [29]), and decentralised PKI (Public Key Infrastructure), such as Certcoin [30].

Application Layer: The application layer (in Figure 2) defines the semantic interpretation of a blockchain system. An example of a semantic interpretation would be to define a crypto-currency and then set up protocols for how such a currency can be exchanged between different entities. Another example is to establish protocols to maintain a state machine embodying programming capabilities within the blockchain, which can be exploited to create and deploy immutable code (the so-called *smart contract*). The application also defines the rewarding mechanism, if any, in the blockchain system.

Consensus Layer: The consensus layer, as presented in Figure 2, is responsible for providing the distributed consensus mechanism in the blockchain that essentially governs the order of the blocks. A critical component of this layer is the proof protocol (e.g., proof of work and proof of stake) that is used to verify every single block, which ultimately is used to achieve the required consensus in the system.

Network Layer: The components in the network layer are responsible for handling network functionalities which include joining in the underlying P2P network, remaining

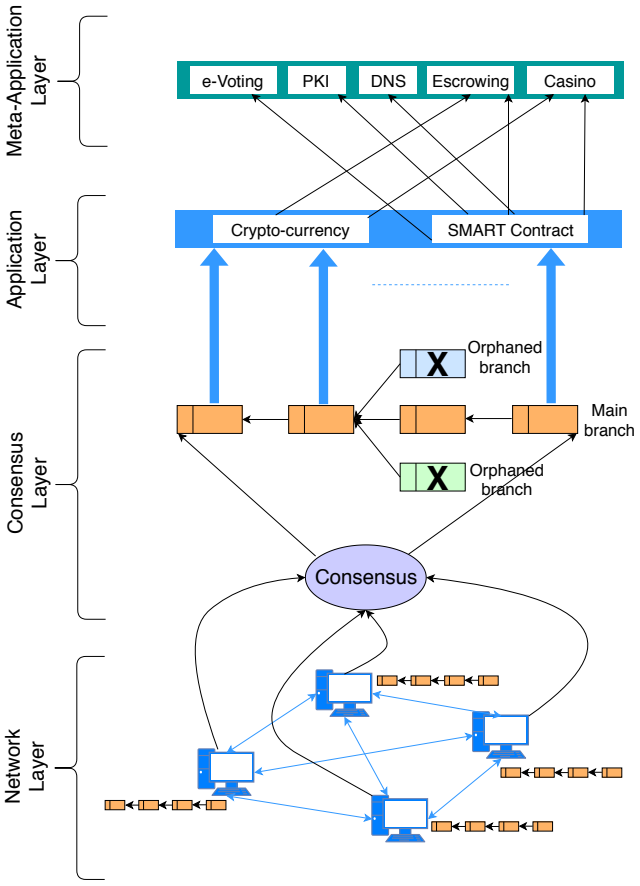


Figure 2: Blockchain Layers

in the network by following the underlying networking protocol, disseminating the current state of the blockchain to newly joined nodes, propagating and receiving transactions and blocks and so on.

4 Consensus Taxonomy & Properties

With the introduction and advancement of different blockchain systems, there has been a renewed interest in distributed consensus with the consequent innovation of different types of consensus algorithms. These consensus algorithms have different characteristics and functionalities. In this section, we first distinguish between two major types of consensus and then present a taxonomy of their properties. Later, in Section 5 and 6, we explore numerous crypto currencies and discuss incentivised consensus algorithms. Similarly, we focus on non-incentivised consensus and the blockchain applications in Section 7.

Consensus mechanisms used by the various blockchain systems can be classified based on the reward mechanism that participating nodes might receive. Therefore, we first classify the consensus mechanisms in blockchain systems into two categories: incentivised and non-incentivised algorithms.

Incentivised Consensus. Some consensus algorithms reward participating nodes for creating and adding a new block in the blockchain. Such algorithms belong to this category. These algorithms are exclusively used in public blockchain systems and the reward provided acts as an incentive for participating nodes to behave accordingly and to follow the corresponding consensus protocol rigorously.

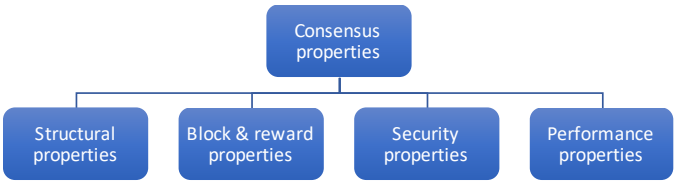


Figure 3: Taxonomy of consensus properties.

Non-incentivised Consensus. Private blockchain systems deploy a type of consensus algorithms that do not rely on any incentive mechanism for the participating nodes to create and add a new block in the blockchain. Such algorithms belong to this category. With the absence of any reward mechanism, these nodes are considered trusted as only authorised (allowed) nodes can participate in the block creation process of the consensus algorithm.

4.1 Consensus properties

Each consensus algorithm has different characteristics and serves different purposes. To compare these disparate groups of consensus algorithms, we need to define evaluation criteria. In this section, we present this evaluation criteria in the form of taxonomies of consensus properties. These properties have been collected from existing researches, such as [5], [4], and compiled as a taxonomy in this work.

The taxonomy is presented in Figure 3. According to this taxonomy, a consensus mechanism has four major groups of properties: *Structural*, *Block & reward* , *Security* and *Performance* properties. Each of these properties is briefly discussed below.

4.1.1 *Structural properties*

Structural properties define how different nodes within a blockchain network are structured to participate in a consensus algorithm. These properties can be sub-divided into different categories as illustrated in Figure 4. We briefly describe each of these categories below.

- **Node types:** It refers to *different types of nodes* that a consensus algorithm is required to engage with to achieve its consensus. The types will depend on the consensus algorithm which will be presented in the subsequent section.
- **Structure type:** It refers to *the ways different nodes are structured* within the consensus algorithm using the concept of a committee. The committee itself can be of two types: single and multiple committees. Each of these committees is described below.
- **Underlying mechanism:** It refers to the *specific mechanism that a consensus algorithm deploys to select a particular node*. The mechanism can utilise lottery, the age of a particular coin or a voting mechanism. A lottery can

Figure 4: Taxonomy of structural properties.

utilise either a cryptography based probabilistic mechanism or other randomised mechanisms. In a voting mechanism, voting can be carried out either in a single or multiple rounds. The coin-age, on the other hand, utilises a special property, which depends on how long a particular coin has been owned by its owner.

Next, we explore different types of voting committees for existing consensus algorithms.

Single committee. A single committee refers to a special group of nodes among the participating nodes which actively participate in the consensus process by producing blocks and extending the blockchain. Each single committee can have different properties. Next, we briefly explore these properties.

- **Committee type:** A committee can be open or close. A committee is open if it is *open* to any participating nodes or closed if it is restricted to a specific group of nodes.
- **Committee formation:** A committee can be formed either implicitly or explicitly. An implicit formation does not require the participating nodes to follow any additional protocol rules to be in the committee, whereas an explicit formation requires a node to follow additional protocol steps to be a part of the committee.
- **Committee configuration:** A committee can be configured in a static or a dynamic fashion.
  - **Static:** In a static configuration, the members of the committee are pre-selected and fixed. No new members can join and participate in the consensus process.
  - **Dynamic:** In a dynamic configuration, the committee members are defined for a time-frame (known as epoch), after which new members are added, and old members are removed based on certain sets of criteria. In such a committee, nodes are selected using a voting mechanism where voting is carried either in a single or multiple rounds. Some consensus algorithms, however, do not specify any specific time-frame, and hence, members can join or leave any time at will. Nodes in such configuration are selected using a lottery mechanism which utilises either a cryptography based probabilistic mechanism or other randomised mechanisms.

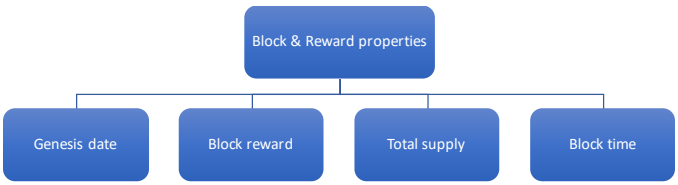


Figure 5: Taxonomy of block & reward properties.

Multiple committee. It has been observed that the time it takes to achieve consensus in a single committee tends to increase as the number of the member starts to increase [5], thereby reducing performance. To alleviate this problem, the concept of multiple committee has been introduced, where each committee consists of different validators [5]. A multiple committee can have different properties. Next, we explore two properties.

- **Topology:** It refers to the way different committees are organised. For example, the topology can be *flat* to indicate that different committees are at the same level or can be *hierarchical* where the committees can be considered in multiple layered levels.
- **Committee configuration:** In addition, like a single committee, the multiple committees can be configured in a static or dynamic way.

4.1.2 Block & reward properties

Properties under this category can be utilised as quantitative metrics to differentiate different crypto-currencies. The properties are (Figure 5): genesis date, block reward, total supply, formula, and block creation time. These properties do not necessarily characterise different consensus algorithm directly, however, most of them (except the genesis date) have a direct and indirect impact on how consensus is achieved in a particular crypto-currency based blockchain system. For example, block reward incentivises miners to act accordingly by solving a cryptographic puzzle, which is then ultimately used to achieve consensus. The properties are described below:

- **Genesis date** represents the timestamp when the very first block was created for a particular crypto-currency.
- **Block reward** represents the reward a node receives for creating a new block.
- **Total supply** represents the total supply of a crypto-currency.
- **Block time** represents the average block creation time of a crypto-currency.

4.1.3 Security properties

A consensus algorithm must satisfy a number of security properties as shown in (Figure 6) and are described below:

- **Authentication:** This implies if nodes participating in a consensus protocol need to be properly verified/authenticated.

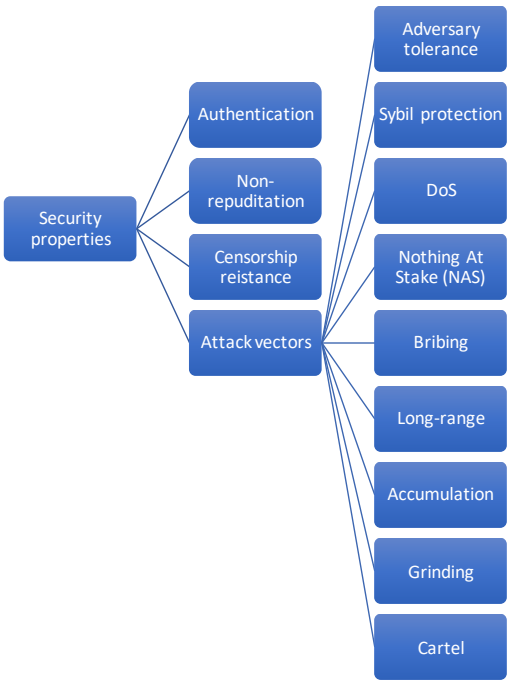


Figure 6: Taxonomy of security properties.

- **Non-repudiation:** This signifies if a consensus protocol satisfies non-repudiation.
- **Censorship resistance:** This implies if the corresponding algorithm can withstand against any censorship resistance.
- **Attack vectors**: This property implies the attack vectors applicable to a consensus mechanism. Here, we present a set of attack vectors that are applicable to any consensus algorithm. The other attack vectors presented in Figure 6 are applicable to a specific class of consensus algorithm. Therefore, we will discuss them in the upcoming sections, when we explore such algorithms.
  - **Adversary tolerance:** This signifies the maximum byzantine nodes supported/tolerated by the respective protocol.
  - **Sybil protection:** In a Sybil attack [34], an attacker can duplicate his identity as required in order to achieve illicit advantages. Within a blockchain system, a sybil attack implicates the scenario when an adversary can create/control as many nodes as required within the underlying P2P network to exert influence on the distributed consensus algorithm and to taint its outcome in her favour.
  - **DoS (Denial of Service) resistance:** This implies if the consensus protocol has any built-in mechanism against DoS attacks.

4.1.4 *Performance properties*

The properties belonging to this group can be utilised to measure the quantitative performance of a consensus protocol. A brief description of each property is presented below with its illustration in Figure 7

- **Fault tolerance:** signifies the maximum faulty nodes the respective consensus protocol can tolerate.

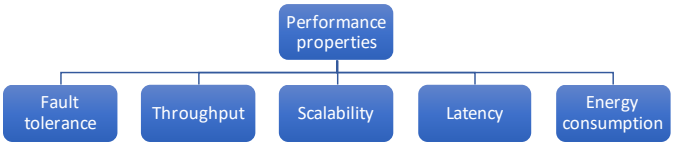


Figure 7: Taxonomy of performance properties.

- **Throughput:** implies the number of transactions the protocol can process in one second.
- **Scalability:** refers to the ability to grow in size and functionalities with- out degrading the performance of the original system [31].
- **Latency (Finality):** refers to *"the time it takes from when a transaction is proposed until consensus has been reached on it"* [5]. It is also known as finality.
- **Energy consumption:** indicates if the algorithm (or the utilising system) consumes a significant amount of energy.

5 Incentivised Consensus: PoW & PoS

In this section, we explore different incentivised consensus algorithms. Such algorithms can be grouped in three major categories: Proof of Work (PoW), Proof of Stake (PoS), and Hybrid Consensus. Among them, this section discusses PoW and PoS algorithms in Section 5.1 and Section 5.2 respectively. For readability, hybrid algorithms are presented in Section 6.

5.1 Proof of Work (PoW)

A Proof of Work (PoW) mechanism involves two different parties (nodes): prover (requestor) and verifier (provider). The prover performs a resource-intensive computational task intending to achieve a goal and presents it to a verifier or a set of verifiers for validation that requires significantly less resource. The core idea is that the asymmetry, in terms of resource required, between the proof generation and validation acts intrinsically as a deterrent measure against any system abuse.

Within this aim, the idea of PoW was first presented by Dwork and Naor in their seminal article in 1993 [33]. They put forward the idea of use PoW to combat email spamming. According to their proposal, an email sender would be required to solve a resource-intensive mathematical puzzle and attach the solution within the email as a proof that the task has been performed. The email receiver would accept an email only if the solution can be successfully verified.

Within the blockchain setting, a similar concept has been adopted. Each PoW mechanism is bound to a threshold, known as the difficulty parameter in many blockchain systems. The prover would carry out the computational task in several rounds until a PoW is generated that matches the required threshold, and every single round is known as a single proof attempt.

PoW has been the most widely-used mechanism to achieve a distributed consensus among the participants

regarding the block order and the chain state. In particular, a PoW mechanism in a blockchain serves two critical purposes:

- A deterrent mechanism against the *Sybil Attack*. In PoW, every mining node would require a significant monetary investment to engage in a resource-intensive PoW mechanism during the block creation process. To launch a Sybil attack, the monetary investment of an attacker will be proportional to the number of Sybil identities, which might outweigh any advantage gained from launching a Sybil attack.
- The PoW mechanism is used as an input to a function which ultimately is used to achieve the required distributed consensus when a fork happens in a blockchain [44].

We differentiate between three major classes of PoW consensus mechanisms: *Compute-bound* PoW, *Memory-bound* PoW and *Chained* PoW. Each of these is explored in the following sections.

5.1.1 Compute-bound PoW

A *Compute-bound PoW*, also known as *CPU-bound PoW*, employs a CPU-intensive function that carries out the required computational task by leveraging the capabilities of the processing units (e.g., CPU/GPU), without relying on the main memory of the system. These particular characteristics facilitate the scenario in which the computation can be massively optimised for faster calculation using Application-specific Integrated Circuit (ASIC) rigs. This has drawn criticisms among the crypto-currency enthusiasts as general people cannot participate in the mining process with their general purpose computers, and the mining process is mostly centralised among a group of mining nodes.

Hashcash by Back et al. [45] is the earliest example to leverage a PoW mechanism in practical systems. Similar to the proposal of Dwork and Naor in [33], Hashcash is also designed to combat spams. In this scheme, the email sender would require to generate a SHA-1 hash with a certain property using as the input a number of information including recipient's email address and date. The property dictates that the generated hash must have at least 20 bits of leading zeroes. Generating an SHA-1 hash with this property would require the senders to engage in several proof attempts in a pseudo-random fashion. Once the hash is generated, it is added within the email header. The verification on the recipient's side is rather trivial, which requires comparing a newly generated hash using the required information with the supplied hash. If they match, it proves that the email sender has engaged in the required amount of computational work. The effectiveness of this approach of fighting spams depends on the hypothesis that spammers rely on the revenue model requiring a mere amount of cost to send a single email. When they would need to engage in such a computationally intensive task for sending every single email, the aggregated associated cost might heavily affect their profit margin and thus deter them from spamming.

Nakamoto consensus is the compute-bound PoW consensus algorithm leveraged in Bitcoin. It is based on the approach of Hashcash, modified to be applied within the blockchain setting. As discussed in Section 3.1, all mining nodes (miners) compete with each other to generate a valid block by finding a solution smaller than the difficulty target. Similar to the idea of HashCash, the miners need to engage in several proof attempts, until the solution is found. In each of these proof attempts, each miner generates a hash using either the SHA-256 or SHA-256d (a double hashing mechanism using SHA-256) algorithm and checks if the generated hash is smaller than the difficulty target. The effect of this distributed engagement is that forks happen, and then the Nakamoto consensus algorithm is utilised to resolve the fork and to achieve a network-wide distributed consensus. The reader is referred back to Section 3.1 (and Figure 1) for a brief description of Nakamoto consensus.

Currently, there are many crypto-currencies that utilise the Nakamoto consensus algorithm. Table 3 shows the top 10 of such currencies according to their market capitalisation as rated by CoinGecko [1] (a website which tracks different activities related to crypto-currencies) as of July 24, 2019. The table also presents their Block and reward properties as presented in Figure 5. It is to be noted that information regarding the properties in Table 3 for these (and other subsequent) currencies has been collected by consulting their corresponding whitepapers, websites and introductory announcements on Reddit website [2].

5.1.2 Memory-bound PoW

To counteract the major criticism of compute-bound PoWs allowing the utilisation of ASIC-based rigs for the mining purpose (see Section 5.1.1), memory-bound PoWs have been proposed. A memory-bound PoW requires the algorithm to access the main memory several times and thus ultimately binds the performance of the algorithm within the limit of access latency and/or bandwidth as well as the size of memory. This restricts ASIC rigs based on a memory-bound PoW to have the manifold performance advantage over their CPU/GPU based counterparts. In addition, the profit margin of developing ASIC with memory and then building mining rigs with them is not viable as of now for these classes of PoWs. Because of these, memory-bound PoWs are advocated as a superior replacement for compute-bound PoWs in de-monopolising mining concentrations around some central mining nodes.

There is a large variety of consensus algorithms belonging to this class, unlike the consensus algorithms of compute-bound PoW which are largely based on Hashcash. These algorithms can be further categorised as follows: Cryptonight; Scrypt and its variants; Equihash; Ethhash/Dagger; Neoscript; and Timetravel. We now describe each of these different types of memory-bound PoW consensus algorithm.

1) CRYPTONIGHT. Cryptonight is a class of PoW consensus algorithms that, in principle, is a memory-hard hash function [32]. It utilises the Keccak hashing function [46] internally and relies on a 2MB scratchpad residing on the memory of a computer. The scratchpad is extensively used to perform numerous read/write operations at pseudo-random

1. https://www.coingecko.com/
2. https://www.reddit.com/

Table 3: Top ten crypto-currencies that utlise Nakamoto consensus algorithm.

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Bitcoin/Bitcoin Cash [69] [70]	03.01.2009	12.5	21	10m
Syscoin [71]	16.08.2014	80.04659537	888	1m
Peercoin [72]	19.08.2012	55.17265345	2000	10m
Counterparty [73]	01.02.2014	All currency in circulation	2.6m	-
Emercoin [75]	11.12.2013	Smooth emission	41	10m
Namecoin [76]	19.04.2011	12.50000000	21	10m
Steem Dollars [77]	04.06.2016	Smooth emission	Unlimited	3s
Crown [78]	08.09.2014	1.8	42	1m
XP	24.08.2016		2220	NA
Omni (Mastercoin) [79]	31.07.2013	16.71249999 Omni	0.6	20s

addresses within that scratchpad. In the final step, the desired hash is generated by hashing the entire scratchpad.

Its reliance on a large scratchpad on the memory of a system makes it resistant towards FPGA and ASIC mining as the economic incentive to create FPGA, and ASIC mining hardware might be too low for the time being. As such, Cryptonight introduces the notion of so called *Egalitarian proof of work [32]* or proof of equality, which enables anyone to join in the mining process using any modern CPU and GPU.

One prominent property of the coins belonging to this class is that all of them support stronger sender-receiver privacy by facilitating anonymous transactions.

Current currencies utilising Cryptonight according to Coingeko as of July 24, 2019 is presented in Table 4. Like Table 3, Table 4 also presents their Block and reward properties as presented in Figure 5.

2) Scrypt and its variants. Scrypt is a password based key driving function (KDF) that is currently used in many crypto-currencies [47]. A KDF is primarily used to generate one or more secret values from another secret key and is widely used in password hashing. Previous key deriving functions such as DES-based UNIX Crypt-function, FreeBSD MD5 crypt, Public-Key Cryptography Standards#5 (PKCS#5), and PBKDF2 do not impose any specific hardware requirements. This enables any attacker launch attacks against those functions using specific FPGA or ASIC enabled hardware, the so-called *custom hardware attacks* [48]. Scrypt has been designed to counteract this threat.

Toward this aim, one of the core characteristics of Scrypt is its reliance on the vast memory of a system, making it difficult to perform using FPGA and ASIC enabled custom hardware. In the underneath, Scrypt utilises Salsa20/8 Core [49] as its internal hash function. A simplified version of Scrypt is used in the corresponding crypto-currencies, which is much faster and easier to implement, and can be performed using any modern CPU and GPU. Hence, anyone can join in the mining process for crypto-currencies using this function. However, the ever-increasing price of crypto-currencies has incentivised miners to produce custom ASIC hardware for some crypto-currencies utilising Scrypt in recent times. An example of such hardware that can be used to mine different Scrypt crypto-currencies is Antminer L3+ [50].

To tackle this issue of exploiting ASIC for mining, several Scrypt variants have been proposed: Scrypt-N/Scrypt Jane/Scrypt Chacha and Scrypt-OG, each providing particular advantages over others. Scrypt-N and Scrypt Chacha rely on SHA256 and ChaCha [52] as their internal hash functions, respectively, whereas Scrypt Jane utilises a combination of different hash functions. All of them support progressive and tunable memory requirements, which can be adjusted after a certain period. This is to ensure that custom ASIC hardware is rendered obsolete once the memory requirement is changed. Finally, Scrypt-OG (Optimised for GPU) is optimised to be eight times less memory intensive than Scrypt [51].

Table 5 shows the top 10 currencies, which either use Scrypt or one of its variants, as per their market capitalisation according to CoinGecko as of July 24, 2019.

3) Equihash Equihash is one of the recent PoW algorithms that has been well received in the blockchain community [55]. It is a memory-bound PoW that requires to find a solution for the Generalised Birthday problem using Wagner's algorithm [56]. Equihash has been designed to decentralise the mining procedure itself, similar to other memory-bound approaches. However, so far, very small portions of such algorithms have succeeded. One of the crucial reasons for this is that their underlying time-memory complexity trade-off is largely constant. This means that reducing memory requirement in these algorithms have little effect on their corresponding time complexity.

Wagner's solution has a steep time-memory complexity trade-off, reducing memory increases time complexity substantially. This premise has been exploited by Equihash to ensure that mining is exclusively proportional to the amount of memory a miner has. Thus, it is more suitable for a general purpose computer, rather than any ASIC-enabled hardware which can only have relatively small memory in order to make their production profitable for the mining process. Due to this reason, it has been claimed that Equihash can support ASIC resistance, at least for the foreseeable future. In addition, the verification is extremely lightweight and even can be carried out in resource-constrained mobile devices. Table 6 shows the eight currencies which utilise Equihash according to CoinGecko as of July 24, 2019.

4) Ethash (Dagger-Hashimoto)/Dagger. Ethash is a memory-bound PoW algorithm introduced for Ethereum with the goal to be ASIC-resistant for a long period of time [58]. It was previously known as Dagger-Hashimoto algorithm [59] because of its utilisation of two different algorithms: Dagger [60] and Hashimoto [60].

Table 4: Top ten crypto-currencies that utilise Cryptonight, with Bytecoin being the first to use this algorithm.

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Monero	18.04.2014	4.86930501	Starting at $M = 2^{64} - 1$ infinite supply	2.0m
Bytecoin	04.07.2012	666.76	184.46 billion	2.0m
Aeon	06.06.2014	5.48	Starting at $M = 2^{64} - 1$, infinite supply.	4.0m
Boolberry	17.05.2014	4.85	18.5 Million	2.0
Karbowanec	30.05.2016.	8.83	Starting with 10 Million, infinite supple	4.0m
Fantomcoin	06.05.2014	smooth emission, 50% coins will be emitted in 6 years and block reward decreases with a similar Starting at $M = 2^{64} - 1$	infinite supply	1.0m
Dashcoin fork of Bytecoin	05.07.2014	1.55		2.0m
QuazarCoin	08.05.2014	smooth emission		2.0
BipCoin	20.08.2016	smooth emission		2.0
Cannabis Industry Coin	16.10.2016	70.00000000	21 M	2.0

Table 5: Top ten crypto-currencies using Scrypt.

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Litecoin [80]	13.10.2011	25.00	84 million	2.5m
Verge [82]	15.02.2016	730.00	16.5 billion	0.5m
Bitmark [83]	13.07.2014	(no longer monitored after 2016)	27.58 million	2.0m
Dogecoin [84]	06.12.2013	10000.00	Total supply	NA
GameCredits [85]	01.06.2015	fixed (12.5 coins)	84 million	1.5m
Einsteinium [86]	01.03.2014	2	2.9 billion	1.0m
Voxels [87]	03.11.2015	smooth emission	2.1 billion	2.5m
Viacoin [88]	18.07.2014	0.63	23 million	0.5m
Hempcoin [89]	9.03.2014	smooth emission	2.5 billion	1.0m

Table 6: Crypto-currencies utilising Equihash algorithm.

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Zcash [90]	28.10.2016	10	21 million	2.5m
Bitcoin Gold [91]	24.10.2017	12.5	21 million	10m
Komodo [92]	15.10.2016	3	200 million	1m
Zclassic [93]	6.11.2016	12.5	21 million	2.5m
ZenCash [94]	30.05.2017	7.5	21 million	2.5m
Hush [95]	Genesis date	11.25	21 million	2.5m
BitcoinZ [96]	10.09.2017	12500.00	21 billion	2.5m
VoteCoin [97]	31.08.2017	125	2.2 billion	2.5m

Dagger is one of the earliest proposed memory-bound PoW algorithm which utilises Directed Acyclic Graph (DAG) for memory-hard puzzle solving with trivial verification that requires less memory to be used in resource constrained devices. However, the Dagger algorithm is proven to be vulnerable towards a shared memory hardware acceleration attack, as discussed in [61]. That is why it has been dropped as a PoW candidate for Ethereum. Hashimoto algorithm, on the other hand, relies on the delay incurred for reading data from memory as the limiting factor and thus, is known as an I-O bound algorithm.

Ethash combines these two algorithms to be ASIC-resistant and functions as follows. Ethash depends on a large pseudo-random dataset, which is recomputed during each epoch. Each epoch is determined by the time it takes to generate 30,000 blocks in approximately five days. This dataset is essentially a directed acyclic graph and hence, is called DAG. During the DAG generation process, a seed is generated at first, which relies on the length of the chain. The seed is then used to compute a 16 MB pseudo-random cache. Then, each item of the DAG is generated by utilising a certain number of items from the pseudo-random cache. This entire process enables the DAG to grow linearly with the growth of the chain. Then, the latest block header and the current candidate nonce are hashed using Keccak (SHA-3) hash function, and the resultant hash is mixed and hashed several times with data from the DAG. The final hashed digest is compared to the difficulty target and accepted or discarded accordingly.

Every mix operation in Ethash requires to have a read in a pseudo-random fashion from the DAG, which is randomly accessed from the memory. This serves two purposes:

- The read operation is limited by the speed of the memory access bandwidth, which is thought to be

theoretically optimal, and thus, more optimisation is less likely.
- Even though the mixing circuitry can be built within an ASIC, the bottleneck would still be the memory access delay.

That is why Ethash is thought to be suitable for use on commodity computing capacity with good powerful GPUs. To achieve the same level of performance, an ASIC would require to accommodate as large memory as a general purpose computer providing a financial disincentive.

There are currently two currencies utilising Ethereum according to coingecko as of July 24, 2019 [62]. Even though Dagger algorithm is proven not to be ASIC resistant, it is being used in 6 currencies [63]. All of these are presented in Table 7.

5) NEOSCRYPT. NeoScrypt, an extension of Scrypt, is a key derivation function that aims to increase the security and performance on CPUs and GPUs while being strong ASIC resistant [146]. Internally it utilises a combination of Salsa 20/20 [49] and ChaCha 20/20 [52] along with Blake2s [74]. Its constructions impose larger memory segment size, and hence, larger temporal buffer requirements. This makes it 1.25 times more memory intensive than Scrypt. The motivation is that this higher requirement of memory will act as a deterrent towards building ASICs for NeoScrypt.

Currently, there are 10 currencies utilising NeoScrypt according to Coingecko as of 18 July, 2019 [147] which are presented in Table 8.

5.1.3 Chained PoW

A chained PoW utilises several hashing functions chained together in a series of consecutive steps. Its main motivation is to ensure ASIC resistance, which is achieved by the underlying mechanisms by which the corresponding hashing functions are chained together. In addition to this, the PoW algorithms belonging to this series aim to address one particular weakness of any compute-bound and memory-bound PoW algorithm: their reliance on a single hashing function. With the advent of quantum computing, the security of a respective hashing algorithm might be adversely affected, which undermines the security of the corresponding blockchain system. If this happens, the old algorithm needs to be discarded, and a new quantum resistant hashing algorithm needs to be incorporated to the respective blockchain using a mechanism called hard-fork. A hard-fork is a mechanism by which a major update is enforced in a blockchain system. This is quite a disruptive procedure that has negative effect on any blockchain system. In such scenarios, a chained PoW algorithm would continue to function until all its hashing functions are broken.

There are several chained PoW algorithms that are currently available.

1) X11/X13/X15. X11 is a widely-used hashing algorithm in many crypto-currencies. In X11, eleven hashing algorithms are consecutively carried our one after another. The hashing algorithms are *blake, bmw, groestl, jh, keccak, skein, luffa, cubehash, shavite, simd, and echo*.

One advantage of X11 is that it is highly energy efficient: GPUs computing X11 algorithm requires approximately 30% less wattage and remains $30 - 50\%$ cooler in comparison to Scrypt [54]. Even though the algorithm has been designed in such a way that it can only be used with CPUs and GPUs, the economic incentives have allowed the creation of ASIC to be used during the mining process.

It has different variants where the number of chained hashing functions differs. For example, X13 utilises 13 hashing functions, and X15 utilises 15 hashing functions.

Table 9 presents the top 10 crypto-currencies utilising these three algorithms, as per their market capitalisation as of July 24, 2019 according to CoinGecko.

2) QUARK. Quark PoW algorithm relies on six different hashing functions: BLAKE [74], Blue Midnight Wish [64], Grøstl [65], [140], JH [66], Keccak and Skein [67]. These functions are implemented in mixed series with nine steps [138]. Within these nine steps, three functions are randomly applied in three steps depending on the value of a bit. The main motivations of mixing these six functions in nine steps are as follows:

- To alleviate the risk of a compromised system in light of its underlying single hashing algorithm being broken.
- To impose restrictions so that Quark can only be mined using CPUs while making it difficult to mine using GPUs and ASICs, because of the usage of ASIC-resistant mechanisms (e.g. Keccak).

However, it did not take long before ASIC mining hardware for Quark appeared in the market, so that this could be mined using a GPU and ASIC [68]. However, the profitability and performance of such hardware are not obvious.

The currencies utilising Quark according to CoinGecko as per July 24, 2019 [139] are presented in Table 10.

3) LYRA2RE. Lyra2RE is a class of chained PoW which utilises five hash functions: BLAKE, Keccak, Lyra2,[13] Skein, and Grøstl. It has been developed by the developers of Vertcoin, a currency based on Lyra2RE. It was designed to be CPU friendly, however, it was discovered in 2015 that the majority of the hashing power utilised for mining VertCoin in its network was facilitated by a botnet stealing CPU cycles from a large number of infected computers. This motivated the Vertcoin developers to release Lyra2REv2, which utilises six hash functions, BLAKE, Keccak, CubeHash, Lyra2, Skein, and Blue Midnight Wish with GPU only PoW. Currently, there are only three currencies utilising Lyra2REv2 according to CoinGecko as of 31 December 2017 which are presented in Table 11.

4) MAGNIFICENT 7. *Magnificent 7* (M7) is a class of chained PoW which utilises seven hash functions to generate the candidate hash during the mining process of Cryptonite coin (not to be confused with the Cryptonight PoW algorithm) [143]. The utilised hash functions are SHA-256, SHA-512, Keccak, RIPEMD, HAVAL, Tiger and Whirlpool. Internally, the header of the candidate block sequentially hashed by the corresponding functions and then multiplied to generate the final hash, which is then compared against the difficulty threshold. Even though it a not memory-bound PoW, it has been claimed that the multiplication operation enables it to run on a general purpose CPU easily, however, makes it difficult to run on GPUs and ASICs [143]. Even so, there are is at least one GPU miner available

Table 7: Crypto-currencies utilising Ethash algorithm. The block rewards are in the corresponding currencies.

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Ethereum [98]	30.07.2015	2	infinite supply	10-20s
Ethereum Classic [99]	30.07.2015		3.88	10-20s
Ubiq [100]	28.01.2017	6	NA	88s
Shift [101]	01.08.2015	1	infinite supply	27s
Expanse [102]	13.09.2015	4	31.4 Million	1.0m
DubaiCoin-DBIX [103]	27.03.2017	6	Total supply	1.5m
SOILcoin [104]	03.10.2015	3.0	Total supply	52s
Krypton [105]	15.02.2016	0.25	2.67 Million	1m 44s

Table 8: Crypto-currencies utilising NeoScrypt and Timetravel 10 algorithms. The block rewards are in the corresponding currencies.

Currency	Algorithm	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Red Pulse [106]		17.10.2017	NA	1.36 Billion	NA
Feathercoin [107]	NeoScrypt	16.04.2013	40	336 Million	1.0m
GoByte [108]	NeoScrypt	17.11.2017	3.71	31.8 Million	2.5m
UFO Coin [109]	NeoScrypt	03.01.2014	625	4 Billion	1.5m
Innova [110]	NeoScrypt	19.10.2017	2.64	1.29 Million	2m
Vivo [88]	NeoScrypt	20.08.2017	4.5	1.1 Million	2m18s
Desire [113]	NeoScrypt	Genesis date	10.45	1.17 Million	2.5m
Orbitcoin [114]	NeoScrypt	28.07.2013	0.5	3.77 Million	6.0m
Phoenixcoin [115]	NeoScrypt	08.05.2013	12.5	98 Million	1.5m
Bitcore [116]	Timetravel 10	April 24, 2017	3.13	21 Million	2.5m

Table 9: Crypto-currencies utilising X11/X13 algorithms. The block rewards are in the corresponding currencies.

Currency	Algorithm	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Dash [117]	X11	January 19, 2014	1.55	22 Million	2.5m
Stratis [118]	X13	August 09, 2016	NA	NA	NA
Cloakcoin [119]	X13	Genesis date	496.00	4.5 Million	1.0m
Stealthcoin [120]	X13	July 04, 2014	NA	20.7 Million	1.0m
DeepOnion [121]	X13	July 13, 2017	4	18.9 Million	4m
HTMLcoin [122]	X15	September 12, 2014	NA	90 Billion	1.0m
Regalcoin [123]	X11	September 28, 2017	NA	7.2 Million	NA
Memetic [124]	X11	March 05, 2016	NA	NA	NA
ExclusiveCoin [125]	X11	June 12, 2016	NA	NA	NA
Creditbit [126]	X11	November 02, 2015	NA	100 Million	1.0m

Table 10: Crypto-currencies utilising Quark algorithm. The block rewards are in corresponding currencies.

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Quark [67]	July 21, 2013	1	247 Million	0.5s
PIVX [128]	NA	5	NA	1.0m
MonetaryUnit [129]	July 26, 2014	18	1 Quadrillion	0.67m
ALQO [130]	October 30, 2017	3	NA	1m
Bitcloud [131]	August 15, 2017	22.5	200 Million	6.5m
Zurcoin [132]	NA	12.5	NA	0.75m
AmsterdamCoin [133]	November 01, 2015	10	84 Million	1.0m
Animecoin [134]	NA	NA	NA	NA

Table 11: Crypto-currencies utilising Lyra2RE algorithm. The block rewards are in corresponding currencies.

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Vertcoin [135]	January 10, 2014	25	84 Million	2.5m
Monacoin [136]	January 01, 2014	25	105 Million	1.5m
Crypto [137]	April 30, 2015	NA	65.8 Million	0.5m

for M7 [144]. Its performance, though, is not known. The corresponding information for Cryptonite is presented in Table 12.

5.1.4 PoW Limitations

PoW (Nakamoto) consensus algorithm has been widely accoladed for its breakthrough in the distributed consensus

Table 12: Information regarding Cryptonite utilising M7 algorithm.

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply (Million)	Block Time
Cryptonite	July 28, 2014	Dynamic	1.84 Billion	1 Minute

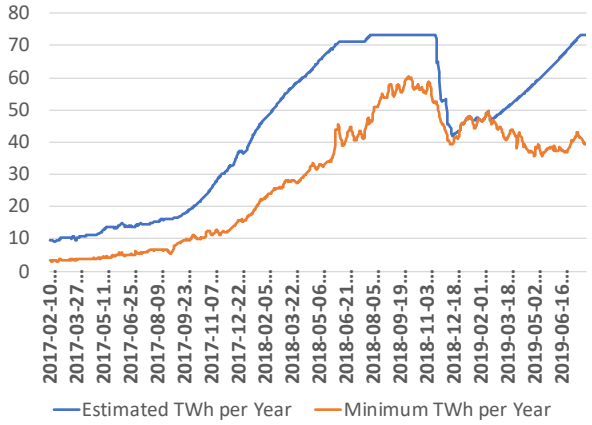


Figure 8: Bitcoin energy consumption over the last years.

paradigm, starting with Bitcoin. It had laid down the foundation for the subsequent advancement, which resulted in different PoW algorithms and crypto-currencies as discussed in the earlier sections. Even so, there are some significant limitations. Next, we briefly discuss these limitations:

- **Energy consumption:** Each PoW algorithm needs to consume electricity to compute the hash. As the difficulty of the network starts to increase, so does the energy consumption. The amount of consumed energy is quite significant when calculated over the whole network consisting of ASIC/GPU mining rigs all around the world. Digiconomist [3] website tracks the electricity consumption of Bitcoin and Ethereum. According do it, the energy consumption of Bitcoin and Ethereum are around 40 TWh (Tera-Watt Hour) and 10 TWh, respectively. Their energy consumption graphs for the last one year are presented in Figure 8 [148] and Figure 9 [149].

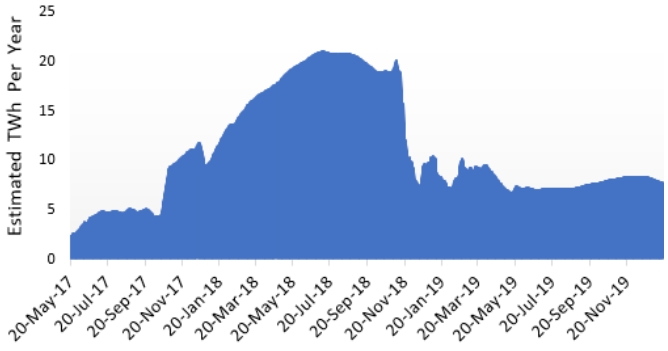


Figure 9: Ethereum energy consumption over the last year.

3. https://digiconomist.net/

To put this into perspective, we present Figure 10, whose data has been collected from [148]. This figure illustrates Bitcoin's energy consumption relative to the electricity consumption of different countries. For example, the electricity consumed by Bitcoin in a year could power up $6,770,506$ American households and is much more than what Czech Republic consumes in a year [148]. The utilisation of this huge amount of electricity has raised the question of sustainability of PoW-based crypto-currencies.

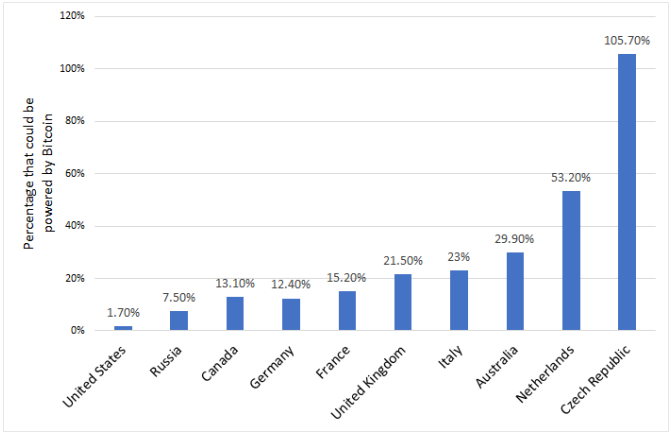


Figure 10: Bitcoin energy consumption relative to different countries.

- **Mining centralisation:** With the ever-increasing difficulty rate, miners within a PoW-based crypto-currency network need to upgrade the capability of their ASIC/GPU mining rigs to increase their chance of creating a new block. Even so, it becomes increasingly difficult for a single miner to join in the mining process without substantial investment in the mining rigs. The consequence is that the *economies of scale* phenomenon strongly impacts the PoW algorithms. The economies of scale in economic theory is the advantage a producer can gain by increasing its output [150]. This happens because the producer can spread the cost of per-unit production over a larger number of goods, which increases the profit margin. This analogy also applies to PoW mining as explained next. A mining pool can be created where the mining resources of different miners are aggregated to increase the chance of creating a new block. Once a mining pool receives a reward for creating the next block, the reward is then proportionally divided among the participating miners. Unfortunately, this has led to the centralisation problems where block creations are limited among a handful of miners. For example, Figure 11 illustrates the distribution of network hashrate among different miners in Bitcoin [152]. As evident from the figure, only five mining pools control the 75% of hashrate of the

whole network. There is a fear that they could collude with each other to launch the 51% attack to destabilise the whole bitcoin network.

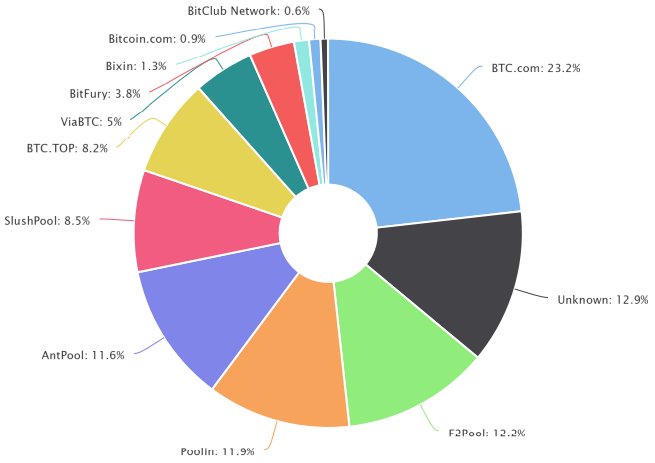


Figure 11: Bitcoin hashrate distribution of mining pools.

- **Tragedy of commons:** Many PoW algorithms suffer an economic problem called the *Tragedy of the commons*. In economic theory, the tragedy of the commons occurs when each entity rushes to maximise its profit from a depleting resource without considering the well-being of all that share the same resource [151]. This situation occurs in a crypto-currency if it is deflationary in nature with limited supply, e.g. Bitcoin. It has been argued when the reward of creating a new block in Bitcoin will reach nearly zero; the miners will have to solely rely on the transaction fees to cover their expenses. This might create an unhealthy competition among the miners to include as many transactions as possible, just to maximise one's profit. The consequence of this is that transaction fees will keep decreasing, which might lead to a situation that miners cannot make enough profit to continue the mining process. Eventually, more and more miners will leave the mining process, which might lead toward 51% attacks or other scenarios that de-stabilise the Bitcoin network.
- **Absence of penalty:** All PoW algorithms (both compute and memory bound) are altruistic in nature in the sense that they reward behaving miners, however, do not penalise a misbehaving miner. One example is that a miner can collude with a group of miners (a phenomenon known as *selfish mining*) to increase its profitability in an illegitimate way [153]. In addition, a miner can engage in Denial-of-Service attack by just not forwarding any transaction or block within the network. Furthermore, such malicious miners can join forces to engage in the *spawn-camping* attack, in which they launch DoS attacks simultaneously over and over again to render the network useless for the corresponding crypto-currency [156]. A penalty mechanism would disincentivize any miner to engage in any type of malicious misbehave.

5.1.5 Analysis

In this section, we summarise the properties of different PoW algorithms in Table 13, Table 14 and Table 15 utilising the taxonomies presented in Section 4. In these tables, a '✓' symbol is utilised to indicate if a particular property is supported by the corresponding algorithm. For other properties, explanatory texts have been used for any particular property.

As presented in Table 13, different types of PoW algorithms share exactly similar characteristics. In these algorithms, they are mainly two types of nodes: clients and miners. Miners are responsible for creating a block using a randomised lottery mechanism. Conversely, clients are the nodes that are responsible for validating each block as well as utilised to transact bitcoin between different users. Committees in these algorithms represent the set of miners, exhibiting the property of a single open committee structure where anyone can join as a miner. The respective committee is formed implicitly in a dynamic fashion, indicating any miner can join or leave whenever they wish.

As per Table 14, none of the algorithms requires any node to be authenticated to participate in the algorithm. All of them have strong support for non-repudiation in the form of digital signature as part of every single transaction. These algorithms have a high level of censorship resistance, which means that it will be difficult for any regulatory agency to impose any censorship on these algorithms. As for the attack vector, each PoW algorithm requires every miner node to invest substantially for mining hardware in order to participate in these consensus algorithms. This feature, thus, acts as a deterrent against any Sybil or DoS attack in any PoW algorithm. The adversary tolerance is based on the assumption that PoW suffers from 51% attacks, and thus, adversary nodes need to have less than 50% of the total hashing power of the network.

According to Table 15, these algorithms have low throughput, and unfortunately, do not scale properly. Furthermore, most of the algorithms require a considerable time to reach finality, and their energy consumption is considerably high, as explained in Section 5.1.4. The fault tolerance in these algorithms is $2f + 1$ like any BFT algorithm, implying they can achieve consensus as long as more than 50% of nodes function correctly.

5.2 Proof of Stake

To counteract the limitations of any PoW algorithm, another type of consensus algorithm, called Proof of Stake (PoS) has been proposed. The earliest proposal of a PoS algorithm can be found on the *bitcointalk* forum in 2011 [154]. Soon after, several projects started experimenting with the idea. Peercoin [72], released in 2012, was the first currency to utilise the PoS consensus algorithm.

The core idea of PoS evolves around the concept that the nodes who would like to participate in the block creation process must prove that they own a certain number of coins at first. Besides, they must lock a certain amount of its currencies, called *stake*, into an escrow account in order to participate in the block creation process. The stake acts as a guarantee that it will behave as per the protocol rules. The node escrows its stake in this manner is known as the

Table 13: Structural properties of PoW consensus algorithms.

Node type	Single committee			Mechanism
	Type	Formation	Configuration	
Clients & Miners	Open	Implicit	Dynamic	Lottery, Randomised

Table 14: Security properties of PoW consensus algorithms.

Authentication	Non-repudiation	Censorship resistance	Attack Vectors		
			Adversary tolerance	Sybil protection	DoS Resistance
x	✓	High	$2f+1$	✓	✓

Table 15: Performance properties of PoW consensus algorithms.

Fault tolerance	Throughput	Scalability	Latency	Energy consumption
$2f+1$	Low	Low	Medium-High	High

stakeholder, leader, forger, or minter in PoS terminology. The minter can lose the stake, in case it misbehaves.

In essence, when a stakeholder escrows its stake, it implicitly becomes a member of an exclusive group. Only a member of this exclusive group can participate in the block creation process. In case the stakeholder gets the chance to create a new block, the stakeholder will be rewarded in one of the two different ways. Either it can collect the transaction fees within the block, or it is provided a certain amount of currencies that act as a type of interest against their stake.

It has been argued that this incentive, coupled with any punitive mechanism, can provide a similar level of security of any PoW algorithm. Moreover, it can offer several other advantages. Next, we explore a few benefits of a PoS mechanism [156].

- **Energy Efficiency:** A PoS algorithm does not require any node to solve a resource-intensive hard cryptographic puzzle. Consequently, such an algorithm is extremely energy efficient compared to their PoW counterpart. Therefore, a crypto-currency leveraging any PoS algorithm is likely to be more sustainable in the long run.
- **Mitigation of Centralization:** A PoS algorithm is less impacted by the economies of scale phenomenon. Since it does not require to build up a mining rig to solve any resource-intensive cryptographic puzzle, there is no way to maximise gain by increasing any output. Therefore, it is less susceptible to the centralisation problem created by the mining pool.
- **Explicit Economic Security:** A carefully designed penalty scheme in a PoS algorithm can deter any misbehaving attack, including spawn-camping. Anyone engaging in such attacks will lose their stake and might be banned from any block creation process in the future, depending on the protocol. This eventually can strengthen the security of the system.

Initial supply: One of the major barriers in a PoS algorithm is how to generate the initial coins and fairly distribute them among the stakeholders so that they can be used as stakes. We term this barrier as the *bootstrap* problem. There are two approaches to address the bootstrap problem:

- Pre-mining: A set of coins are pre-mined, which are then sold before the launch of the system in an IPO (Initial Public Offering) or ICO (Initial Coin Offering).
- PoW-PoS transition: The system starts with a PoW system to fairly distribute the coins among the stakeholders. Then, it slowly transitions towards the PoS system.

Reward process: Another important aspect is the rewarding process to incentivise the stakeholder to take part in the minting process. Unlike any PoW, where a miner is rewarded with new coins for creating a valid block, there is no reward for creating a valid block. Instead, to incentivise a minter, two types of reward mechanisms are available within a PoS algorithm:

- Transaction Fee: The minter can collect fees from the transactions included within the minted block.
- Interest rate: A lower interest rate is configured, which allows the currency to inflate over time. This interest is paid to the minter as a reward for creating a valid block.

Selection process: A crucial factor in any PoS algorithm is how to select the stakeholder who can mint the next block. In a PoW algorithm, a miner is selected based on who can find the resource-intensive desired hash. Since PoS does not rely on hind such a hash as the mechanism to find the next block, there must be a mechanism to select the next stakeholder.

Currently, there three different approaches to Proof of Stake: Chained, BFT, and Delegated.

CHAINED POS. The general idea of a chained PoS is to deploy a combination of PoW and PoS algorithms chained together to achieve any consensus. Because of this, there can be two types of blocks, PoW and PoS blocks, within the same blockchain system. To accomplish this, the corresponding algorithm relies on different approaches to select/assign a particular miner for creating a PoW block or select a set of validators for creating a PoS block in different epochs or after a certain number of blocks created. In general, a chain based PoS can employ any of the following three different approaches to select the miner/stakeholder:

- *Randomised PoW Mining:* A miner who can solve the corresponding cryptographic PoW puzzle is selected in a random fashion.
- *Randomised Stakeholder Selection:* A randomised PoS utilises a probabilistic formula that takes into account the

staked currencies and other parameters to select the next stakeholder. The other parameters ensure that a stakeholder is not selected only based on the number of their staked coins and act as a pseudo-random seed for the probabilistic formula.

- *Coin-age based selection.* A coin-age is defined as the holding period of a coin by its owner. For example, if an owner receives a coin from a sender and holds it for five days then the coin-age of the coin can be defined as five coin-days. Formally,

$$coin - age = coin * holdingperiod$$

Algorithms belonging to this class select the stakeholder using staked coins of the stakeholders and their corresponding coin-age.

In general, a chained PoS algorithm favours towards availability over consistency when network partition occurs, as per the CAP theorem.

BFT PoS. BFT PoS is a multi-round PoS algorithm. In the first step, a set of validators are pseudo-randomly selected to propose a block. However, the consensus regarding committing this block to the chain depends on the $> 2/3$ quorum of super-majority among the validators on several rounds. It inherits the properties of any BFT consensus, and as such, it tolerates up to $1/3$ of byzantine behaviour among the nodes.

In general, a BFT PoS algorithm favours towards consistency over availability when network partition occurs, within the setting of CAP theorem.

Delegated Proof of Stake. Delegated Proof of Stake (or DPoS in short) is a form of consensus algorithm in which reputation scores or other mechanisms are used to select the set of validators [184]. Even though it has the name Proof of Stake associated with it, it is quite different from other PoS algorithms.

In DPoS, users of the network vote to select a group of delegates (or witnesses) who are responsible for creating blocks. Users utilise reputations scores or other mechanisms to choose their delegates. Delegates are the only entities who can propose new blocks. For each round, a leader is selected from the set of delegates who can propose a block. How such a leader is chosen depends on the respective system. The leader gets rewards for creating a new block, and is penalised and de-listed from the set of validators if it misbehaves.

The delegates themselves compete with each other to get included in the validator list. In such, each validator might offer different levels of incentives for the voters who vote for it. For example, if a delegate is selected to propose a block, it might distribute a certain fraction of its reward among the users who have selected it. Since the number of validators is small, the consensus finality can be fast.

Next, we explore several crypto-currencies or mechanisms that use the above mentioned PoS approaches.

5.2.1 Chained PoS

Next, we present two examples of a chained PoS algorithm to illustrate how this approach has been applied in practice.

1) PeerCoin (PPCoin). Peercoin is the first crypto-currency to formalise the notion of PoS by utilising a hybrid PoW-PoS protocol [174]. The Peercoin protocol is based on the assumption that *coin-age* can be leveraged to create a PoS algorithm which is as secure as any PoW algorithm while minimising the disadvantages associated with a PoW algorithm.

Peercoin protocol recognises two different kinds of blocks: PoW blocks and PoS blocks, within the same blockchain. These blocks are created by two separate entities: miners and minters. Miners are responsible for creating PoW blocks, similar to Bitcoin whereas minters are responsible for creating PoS blocks. Irrespective of the last block type, the next block either can be a PoW block or a PoS block, and these entities compete with each other to create the next block [175]. Miners compete with other miners to find a valid PoW block that matches the PoW difficulty target, similar to Bitcoin. Similarly, minters compete among themselves to find a valid PoS block that matches the PoS difficulty target (similar to a PoW algorithm but requires much less computation). As soon as any PoW or PoS block is found, it is broadcast to the network, and other nodes validate it.

Within a PoS block, a minter utilise their holding coins as a stake, and the minter is rewarded approximately 1% per annum based on the coin-age of the staked coins. The reward is paid out for each block in a newly created special transaction called the *coinstake* transaction. Each coinstake transaction consists of the number of staked inputs and a *kernel*, containing the hash that meets the PoS difficulty. The hash itself is calculated over a small space and hence not computationally intensive at all. It utilises the number of staked inputs and a probabilistic variable, whereas the difficulty condition is calculated utilising the coin-age of the staked inputs as well as a difficulty parameter. This parameter is adjusted dynamically to ensure that one block is created in 10 minutes. In other words, the valid kernel depends on the coin-age of the staked inputs, and the higher the coin-age, the higher is the probability to match the difficulty.

The coinstake transaction is paid to the minter, which contains the coins staked along with the reward. Once a PoS block is added to the chain, the coin-age of the staked coins is reset to zero. This indicates that all the stacked coins are consumed. This ensures that the same coins cannot be used over and over again to create a PoS block within a short period of time. The main chain in Peercoin is selected based on the highest total coin-age consumed in all blocks. That means, if a PoW block and PoS block are received simultaneously as the next block by a node, the algorithm dictates the PoS block to be selected over the PoW block.

The block reward for a PoW block in Peercoin decreases and will cease to be significant after a certain period of time. It is currently used for the coin generation and distribution purpose and will be completely phased out in the future [205]. It has no role whatsoever on securing the network, which is largely based on the PoS algorithm. Once the PoW algorithm is phased out, it is suggested that the energy consumption of Peercoin will be significantly low while providing similar security as any PoW algorithm.

Peercoin is highly regarded for formalising the first alternative mechanism to PoW, however, it suffers from all the attack vectors of PoS, as presented in Section 5.2.4. Two

other coins Black and Nxt removes age from the equation in order to avoid the exploitation of the system by the dishonest entities having a significant amount of coins.

2) CASPER FFG. Casper the Friendly Finality Gadget (CFFG) is a PoW-PoS hybrid consensus algorithm proposed to replace the Ethereum's PoW consensus algorithm [181]. In fact, CFFG provides an intermediate PoS overlay on top of its current PoW algorithm so that Ethereum is transformed to a pure PoS protocol called Casper the Friendly Ghost (CTFG) described below (Section 5.2.2).

The PoS layer requires the participation of validators. Any node can become a validator by depositing some Ethereum's native crypto-currency called *Ether* to a designated smart-contract, which acts as a security bond. The network itself will mostly consist of PoW miners who will mine blocks according to its current PoW algorithm. However, the finalisation/check-pointing of blocks will be carried out by PoS validators. The check-pointing/finalisation is the process to ensure that the chain becomes irreversible up to a certain block and thus, short and low range attacks (particular types of PoS only attacks presented in Section 5.2.4) as well as the 51% attack cannot be launched beyond the check-pointing block.

The check-pointing occurs every 50 blocks, and this interval of 50 blocks is called an *epoch* [158]. The finalisation process requires two rounds of voting in two successive epochs. The process is as follows. In an epoch, the validators vote on a certain checkpoint c (a block). A super-majority (denoted as $+2/3$) occurs when more than $2/3$ of the validators vote for the checkpoint c. In such a case, the checkpoint is regarded as *justified*. If in the next epoch, $(+2/3)$ of the validators vote on the next checkpoint c' (a block which is a child of the block belonging to C), c' is considered justified whereas c is considered finalised. A checkpoint created in this manner for each epoch is assumed to create a checkpoint tree where c' is a direct child of c. The process can be summarised in the following way: $+2/3$ Vote $c \rightarrow$ Justify $c \rightarrow +2/3$ Vote $c' \rightarrow$ Finalize c and Justify c'

Once a checkpoint is finalised, the validators are paid. The payment is interest-based and is proportional to the number of ethers deposited. If it occurs that there are two checkpoints, it signifies that a fork has occurred. This can only happen when a validator or a set of validators has deviated from the protocol. In such cases, a penalty mechanism is imposed in which the deposit of the violating validator(s) is destroyed.

In essence, CGGF is a combination of Chained and BFT consensus mechanisms with strong support for availability over consistency. Its properties ensure that block finalisation occurs quickly, and the protocol is mostly secure against all PoS attacks except the cartel formation attack (a particular type for PoS only presented in Section 5.2.4). However, it is to be noted that this consensus mechanism has not been implemented yet. Therefore, it is yet to be seen how it performs in reality.

5.2.2 BFT PoS

In this section we describe three notable BFT PoS algorithms that have had significant uptake in practice: Tendermint, CTFG and Ouroboros.

1) TENDERMINT. Tendermint is the first to showcase how the BFT consensus can be achieved within the PoS setting of blockchain systems [178], [179], [180]. It consists of two major components: a consensus engine known as Tendermint Core and its underlying application interface, called the *Application BlockChain Interface* (ABCI). The Tendermint core is responsible for deploying the consensus algorithm, whereas the ABCI can be utilised to deploy any blockchain application using any programming language.

The consensus algorithm relies on a set of validators. It is a round-based algorithm where a proposer is chosen from a set of validators. In each round , the proposer proposes a new block for the blockchain at the latest height. The proposer itself is selected using a deterministic round-robin algorithm, which ultimately relies on the voting power of the validators. The voting power, on the other hand, is proportional to the security deposit of the validators.

The consensus algorithm consists of three steps (propose, pre-vote, and pre-commit) in each round bound by a timer equally divided among the three steps, thus making it a weakly synchronous protocol. These steps signify the transition of states in each validator. Figure 12 illustrates the state transition diagram for each validator. At the beginning of each round, a new proposer is chosen to propose a new block. The proposed block needs to go through a two-stage voting mechanism before it is committed to the blockchain.

When a validator receives the proposed block, it validates the block at first, and if okay, it pre-votes for the proposed block. If the block is not received within the *propose* timer or the block is invalid, the validator submits a special vote called *Prevote nil*. Then, the validator waits for the *pre-vote* interval to receive pre-votes from the super-majority (denoted as $+2/3$) of the validators. A $+2/3$ pre-votes signifies that the super-majority validators have voted for the proposed block, implying their confidence on the proposed block and is denoted as a *Polka* in Tendermint terminology. At this stage, the validator pre-commits the block. If the validator does not receive enough pre-votes for the proposed block, it submits another special vote called *Precommit nil*. Then, the validator waits for the *pre-commit* time-period to receive $+2/3$ pre-commits from the super-majority of the validators. Once received, it commits the block to the blockchain. If $+2/3$ pre-commits not received within the *pre-commit* time-period, the next round is initiated where a new proposer is selected, and the steps are repeated.

To ensure the safety guarantee of the algorithm, Tendermint is also coupled with locking rules. Once a validator pre-commits a block after a polka is achieved, it must lock itself onto that block. Then, it must obey the following two rules:

- it must pre-vote for the same block in the next round for the same blockchain height,
- the unlocking is possible only when a newer block receives a polka in a later round for the same blockchain height.

With these rules, Tendermint guarantees that the consensus is secure when less than one-third validators exhibit byzantine behaviour, meaning conflicting blocks will never be committed at the same blockchain height. In other words, Tendermint guarantees that no fork will occur under this assumption. Since Tendermint favours safety over availability,

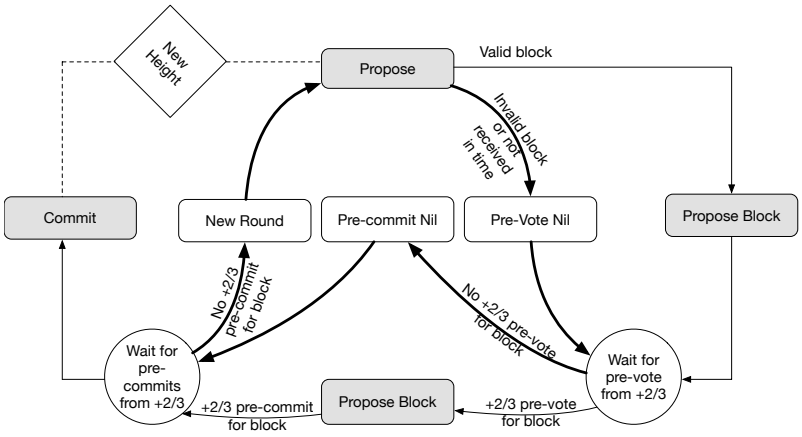


Figure 12: Tendermint consensus steps.

it has one particular weakness. It requires 100% uptime of its $+2/3$ (super-majority) validators. If more than one-third $(+1/3)$ are validators are offline or partitioned, the system will stop functioning [178]. In such cases, out-of-protocol steps are required to tackle this situation.

Unlike PoW or other PoS algorithms that come with defined reward mechanisms and crypto-currency applications, the latest version of Tendermint more likely acts as the consensus plugin, which can be retro-fit to other blockchain systems. For example, Tendermint has been integrated with a private instantiation of Ethereum in a Hyperledger project called Burrows [209]. That is why there is no reward/punishment mechanism defined in Tendermint. However, it can be easily introduced in the application layer via the ABCI. For example, a reward mechanism can be introduced for the proposer and the validator to motivate them to engage in the consensus process. A node can become a validator by bonding a certain amount of security deposit. The deposit is destroyed, in case the corresponding validator misbehaves, and thus acts as a deterrent for the validator to launch any attack in the network. Together with the consensus algorithm and a carefully designed reward and punishment mechanism, all PoS attacks can be effectively handled.

2) Casper the Friendly Ghost (CTFG). CTFG is a pure BFT PoS algorithm that aims to transform Ethereum to a PoS-only blockchain system in the future [182]. As described above, CFFG is geared towards a gentle transition from a PoW to a PoS model for Ethereum, where CTFG will take control of the consensus mechanism ultimately.

CTFG is based upon a rigorous formal model called Correction by Construction (CBC) that utilises the GHOST (Greedy Heaviest-Observed Subtree) primitive as its consensus rule during fork [183]. The idea is that the CTFG protocol will be partially specified at the initial stage along with a set of desired properties. Then, the rest of the protocol is dynamically derived in such a way that it satisfies the desired properties - hence the name correction by construction. This is in contrast to the traditional approach for designing a protocol where a protocol is fully defined at first, and then it is tested to check if it satisfies the desired properties [156].

To achieve this, CTFG introduces a safety oracle, acting as an ideal adversary, which raises exceptions when a fault occurs and also approximates the probability of any future failure. Based on this, the oracle can dynamically fine-tune the protocol as required to evolve it towards its completion.

Similar to CFFG, CTFG also requires a set of bonded validators that will bond ethers as a security deposit in a smart-contract. However, unlike any other PoS mechanisms, the validators will bet on the block, which has the highest probability to be included in the main chain according to their own perspective. If that particular block is included in the main chain, the validators receive rewards for voting in favour of the block. Otherwise, the validators receive certain penalties.

Like any PoW algorithm, CTFG favours availability over consistency. This means that blocks are not finalised instantly, like Tendermint. Instead, as the chain grows and more blocks are added, a previous block is considered implicitly final. A major advantage of CTFG over Tendermint is that it can accommodate dynamic validators. This is because the finality condition in Tendermint requires that its block interval is short, which in turn demands a relatively lower number of pre-determined validators. Since CTFG does not rely on any instant finality, it can theoretically accommodate a higher number of dynamic validators.

CTFG is currently is the most comprehensive proposal which addresses all PoS attack vectors. However, it is to be noted that this is just a proposal at the current stage. Therefore, its performance in real settings is yet to be analysed.

3) Ouroboros. Ouroboros is a provably secure PoS algorithm [185], [186] utilised in the Cardano platform [187]. Cardano is regarded as third-generation blockchain system supporting smart-contract and decentralised application without relying on any PoW consensus algorithm.

In Ouroboros, only a stakeholder can participate in the block minting process. A stakeholder is any node that holds the underlying crypto-currency of the Cardano platform called **Ada**. Ouroboros is based on the concept of *epoch*, which is essentially a predefined time period. Each epoch consists of several slots. A stakeholder is elected for each slot to create a single block, meaning a block is created in each slot. The selected stakeholder is called a slot leader and is elected by a set of *electors*. An elector is a specific type of stakeholder which has a certain amount Ada in its disposal.

In each epoch, the electors select the set of stakeholders for the next epoch using an algorithm called *Follow the*

Satoshi (FTS). The FTS algorithm relies on a random seed to introduce a certain amount of randomness in the election process. A share of the random seed is individually generated by all electors who participate in a multiparty computation protocol. Once the protocol is executed, all electors posses the random seed, constructed with all of their shares. The FTS algorithm utilises the random seed to select a coin for a particular slot. The owner of the coin is then elected as the slot leader. Intuitively, the more coins a stakeholder possesses, the higher is its probability of being selected as the slot leader.

Ouroboros is expected to provide a transaction fee based reward to incentivise stakeholders to participate in the minting process. However, the details are in the process of being finalised. It has been mathematically proven to be secure against almost all PoS attack vectors except the cartel formation [185]. Nevertheless, how it will perform once deployed is yet to be seen.

5.2.3 DPoS

There are several mechanisms deployed by different crypto-currencies under the general category of DPoS. Next, we present a few prominent approaches of some well-known DPoS based crypto-currencies. Our analysis of these crypto-currencies are summarised in Table 16.

1) EOS. EOS is the first and the most widely known DPoS crypto-currency and smart-contract platform as of now [188]. With the promise of greater scalability and higher transactions per second than Ethereum, it raised 4 billion USD in the highest ever ICO event to date [190]. Initial EOS currency was created on the Ethereum platform, and later migrated to their own blockchain network. The DPoS consensus algorithm of EOS utilises 21 validators, also known as *Block Producers* (BPs). These 21 validators are selected with votes from EOS token (currency) holders. The number of times a particular BP is selected to produce a block is proportional to the total votes received from the token holders.

Evey DPoS currency must create an initial supply before the network is operational. This supply is used to select 21 BPs (with voting) as well as to reward the BPs for creating blocks, and thus, securing the network. EOS had an initial supply of 1 Billion EOS tokens with an annual inflation of 5%. Among the inflated currencies, 1% is used to reward the block producers, whereas the rest of the 4% are kept for future R&D for EOS [191]. Currently, an EOS block is created in 0.5s. Blocks in EOS are produced in rounds where each round consists of 21 blocks [192]. At the beginning of each round, 21 BPs are selected. Next, each of them gets a chance to create a block in pseduo-random fashion within that particular round. Once a BP produces a block, other BPs must validate the block and reach into a consensus. A block is confirmed only when $(+2/3)$ majority of the BPs reach the consensus regarding the validity of the block. Once this happens, the block and the associated transactions are regarded as confirmed or final, so no fork can happen.

2) TRON. Tron is another popular DPoS based crypto-currency [193]. With an initial supply of 99 Billion Tron tokens (represented with **TRX**), it is another smart-contract supported blockchain platform, very similar to Ethereum and EOS in functionality. Its consensus mechanism utilises 27 validators, known as Super Representatives (SRs) [194]. The SRs are selected in every six hours with votes by TRX holders who must freeze a certain amount of TRX to vote for an SR. The deposits amount can be frozen back after three days once the voting is cast [195]. A block in Tron is created in every 3s for the corresponding SR receives a reward of 32 TRX. Another important feature of Tron is that there is no in-built inflation mechanism in the protocol, which implies that the total supply will remain constant throughout its lifespan.

3) TEZOS. Tezos is, like EOS and Tron, a smart-contract platform which utilises a variant of DPoS consensus algorithm [196]. With a block reward of 16 XTZ (Tezos currency) and block creation time of 60s, Tezos does not require any pre-defined number of stakeholders (or *Bakers* as defined in Tezos) [197]. This differs Tezos from other DPoS currencies. Instead, the consensus mechanism utilises a dynamic range of stakeholders where anyone holding a substantial amount of XTZ can be a stakeholder. This limits general users to participate in the consensus mechanism. To rectify this problem, Tezos provides a mechanism by which anyone can delegate their XTZ to someone so that it can accumulate the required number of XTZ to be a baker. In return, the baker would return a certain proportion of their received block reward to the delegating party. Tezos started with an initial supply of 765 Million XTZ tokens. It relies on an annual inflation of 5.51% and the inflated currencies are used to reward the bakers.

4) LISK. Lisk is a unique DPoS blockchain platform which, enables the development of DApps using JavaScript [200]. Another unique feature of Lisk is its ability to accommodate and then to operate with multiple blockchains, known as *sidechains* along with a central blockchain called *mainchain*. Each sidechain can be deployed and maintained by a particular application provider, which needs to be synced with the mainchain as per the Lisk's protocol rule. In this way, different applications can leverage different sidechains simultaneously without burdening off the mainchain. Even though the responsibility of maintaining a sidechain relies on the particular application provider, the mainchain must be maintained with the Lisk DPoS consensus protocol, which utilises 101 delegates [201]. Only these delegates can produce a block. These delegates are selected using votes from Lisk currency (denoted with LSK) owners, where each holder has 101 votes. The weight of each vote is proportional to the amount of LSK owned by the respective owner. The selection of delegates happens before a round, where each round consists of 101 block generation cycle. Thus, in a round, each delegate is randomly selected to create a block. It has a block creation time of 10 seconds and block reward of 5 LSK. Started with an initial supply of 100 million LSK, Lisk has a current supply of 132 million with an annual inflation of 5.65%.

5) ARK. Ark is yet another DPoS based blockchain platform [202]. It utilises 51 delegates to create 51 blocks in each round [203]. With a block creation time of 8s, each round lasts for 408s. Each delegate receives 2 ARK (the native currency of the ARK platform) for creating a block. It had

an initial supply of 125 million. With an annual inflation of 5.55, the supply was around 142 million (as of June 2019). Like other DPoS blockchains, the delegates in Ark are also selected with votes by Ark currency owner, where the weight of each is proportional to the amount of ARK owned by the voter.

5.2.4 Limitations of PoS

Even though the variants of different PoS algorithms offer several significant advantages, there are still a few disadvantages in these classes of algorithms. We explore these disadvantages below.

- **Collusion:** If the number of validators is not large enough, it might be easier to launch a 51% attack on the corresponding consensus algorithm by colluding with other validators.
- **Wealth effect:** The sole reliance on coin-wealth in a consensus algorithm or for the selection of validators creates an environment where people with a large portion of coins can exert greater influence.

In addition to these disadvantages, there have been a few other attack vectors identified for the PoS algorithms:

- **Nothing-at-stake (NAS) attack [157]:** During a blockchain fork, an attacker might attempt to add its newly created block in all forked branches to increase their probability to add their block as the valid block. Such scenario is unlikely to occur in any PoW algorithm. This is because a miner would need to share their resources in order to mine at different branches. This would eventually decrease their chance of finding a new block because of the resources shared in multiple branches. Since it does not cost anything for a minter in a PoS algorithm to add blocks in multiple parallel branches, the attacker is motivated to do so. Applying a penalty for such misbehaviour could effectively tackle this problem.
- **Bribing (short-range, SR) attack [157], [176]:** In this attack, an attacker tries to double spend by creating a fork. An example of this attack would be as follows. The attacker pays to a seller to buy a good. The seller waits for a certain number of blocks (e.g., six blocks) before the good is delivered to the attacker. Once delivered, the attacker forks the main chain at the block (e.g., six blocks back, which is relatively short and hence the name) in which the payment was made. Then, the attacker bribes other minters to mint on top of the forked branch. As long as the bribed amount is lower than the price of the delivered good, it is always profitable for the attacker. The colluding minter has nothing to lose if it is coupled with the nothing-at-stake attack on their part but can gain from the bribery. Again, it can be tackled by introducing a penalty mechanism for all misbehaving parties.
- **Long-range (LR) attack [157]:** In this attack, the attacker attempts to build an alternative blockchain starting from the earliest blocks if the attacker can collude with the majority of the stakeholders. The motivation might be similar to double spending or related issues providing advantages to the attacker as well as the colluded stakeholders. As explained above, the colluded stakeholder has nothing to lose if it can be coupled with the nothing-at-stake attacks. Check-pointing is one of the methods by which it can be tackled. The check-pointing codifies a certain length of the blockchain to make it immutable up to that point, and thereby undermining the attack. This is because the attacker cannot fork the blockchain before that check-point.
- **Coin-age accumulation (CAC) attack [157], [176]:** The PoS algorithms that rely on the uncapped coin-age parameter are susceptible to this attack. In this attack, the attacker waits for their coins to accumulate enough coin-age to exploit the algorithm for launching double spends by initiating a fork. This attack can be tackled by introducing a cap on the coin-age which minimises the attack vector.
- **Pre-computing (PreCom) attack [157], [155]:** A pre-computing attack, also known as Stake-grinding attack, would allow an attacker to increase the probability of generating subsequent blocks based on the information of the current block. If there is not enough randomness included in the PoS algorithm, the attacker can attempt to pre-compute subsequent blocks by fine-tuning information of the current block. For a particular set of information (e.g., a set of transactions), if the attacker finds that the probability of minting a few subsequent blocks is less than desired, the attacker can update the set of transactions to increase their probability of determining the next few blocks. It can be effectively tackled by introducing a secure source of randomness in the algorithm.
- **Cartel formation (CAF) attack [158]:** In economic theory, an oligopoly market is dominated by a small set of entities having greater influence or wealth than other entity. They can collude with one another by forming a cartel to control price or reduce competition within the market. It has been argued that "*Blockchain architecture is mechanism design for oligopolistic markets.*" [159] which affects both PoW and PoS algorithms. Such a cartel can launch 51% attacks on the PoS algorithm or exploit the stakes to monopolise the PoS algorithm.

5.2.5 Analysis

In this section, we summarise the properties of different PoS algorithms utilising the taxonomies and PoS attack vectors in Table 17, Table 18, Table 19 and Table 20. Like before, a '✓' symbol has been utilised to indicate if the corresponding algorithm supports a particular property, and the 'X' symbol signifies that the particular property is not supported. The '-' symbol implies that the property is not applicable, whereas the symbol '?' indicates that no information has been found for that particular feature. For other properties, explanatory texts have been used as well.

From Table 17, only chained algorithms are based on multiple committee utilising a flat topology with a dynamic configuration. These algorithms also use a probabilistic lottery to select a minter. Conversely, other PoS algorithms, except Tendermint, are based on the single committee having an open type and explicit formation with a dynamic configuration and mostly rely on voting mechanisms. Tendermint uses a closed committee with a static configuration.

As per Table 18, none of the algorithms, except Tendermint requires any node to be authenticated to participate

Table 16: Comparison of DPoS Currencies with '-' signifying not applicable.

Currency	Genesis date (dd.mm.yyyy)	Initial supply	Inflation	Current supply (23.05.2019)	Block reward	Block Time	Validator nos
EOS	01.07.2017	1 Billion	5%	1.04 Billion	1% of inflated currency divided among 21 validators	0.5s	21
Tron	28.08.2017	99 Billion	-	99 Billion	32 TRX	3s	27
Tezos	30.06.2018	765 Million	5.51%	795 Million	16 XTZ	60s	Not pre-defined
Lisk	24.05.2016	100 Million	5.67%	132 Million	5 LSK	10s	101
Ark	21.03.2017	125 Million	5.55%	142 Million	2 ARK	8s	51

in the algorithm. All of them have strong support for non-repudiation in the form of digital signature as part of every single transaction. These algorithms have a high level of censorship resistance, as do all PoW algorithms. As for the attack vector, each PoS algorithm requires every miner node to invest substantially to participate in this algorithm. This feature, thus, acts as a deterrent against any Sybil or DoS attack in any PoS algorithm. The adversary tolerance for Chained systems can be calculated using this formula: $min(2f+1, 3f+1) = 3f+1$. This is because a chained algorithm utilises both PoW and PoS algorithms and thus needs to consider the adversary tolerance for both of them. We consider the minimum of these two $(3f+1)$. The supported adversary tolerance for other algorithms is $3f+1$ except BFT Ouroboros whose adversary tolerance is $2f+1$.

According to Table 20, all BFT, and DPoS algorithms have considerably high throughput, low latency, and high scalability. Their energy consumption is negligible. However, the chained algorithms have a comparatively lower throughput, lower scalability, and higher latency with respect to their BFT and DPoS counterparts. The fault tolerance of chained and BFT algorithms is $2f+1$ like any BFT algorithm, implying they can achieve consensus as long as more than 50% of nodes function properly. However, DPoS algorithm requires a $3f+1$ fault tolerance.

Table 19 outlines a comparison of additional attack vectors with symbols representing the usual semantics. CTFG, Tentermint, and Ouroboros have mitigation mechanisms against these attack vectors. However, Casper FFG, and any DPoS algorithms cannot successfully defend against the cartel formation attack. Peercoin, on the other hand, has mechanism against this cartel formation attack, unfortunately, suffers from all other attack vectors.

Finally, a comparison of the selected DPoS crypto-currencies is presented in Table 16.

6 Incentivised Consensus: Beyond PoW and PoS

Some consensus algorithms take a different approach in which they do not solely rely on any PoW or PoS mechanism. Instead, they use an approach in which a PoW/-PoS mechanism is combined with another approach. We consider such algorithms as hybrid algorithms which are presented in Section 6.1. Other approaches adopt a more drastic approach in which they do not leverage any type PoW/PoS algorithm whatsoever. Such algorithms are tagged as *N-POS/POW* (to symbolise Non-PoS/PoW) algorithms and discussed in Section 6.2.

6.1 Hybrid Consensus

In this section, we outline a new breed of consensuses algorithms that combine either a PoW or PoS algorithm or both with another novel algorithm or mechanism, thus creating a hybrid mechanism.

1) Proof of Research (PoR). Proof of research is a hybrid approach that combines proof-of-stake with the proof-of-BOINC [160]. BOINC stands for Berkeley Open Infrastructure for Network Computing [162]. It is a grid computing platform widely used by scientific researchers in different domains by allowing them to exploit the idle computing resources of personal computers around the world. With the proof-of-BOINC, a researcher has to prove his contribution for the BOINC research work.

The PoR mechanism is leveraged by Gridcoin [160], [161], a crypto-currency that can be earned by anyone by sharing their computing resources with the BOINC project. The mechanism by which PoS and Proof-of-BOINC are tied together for the PoR is explained next [161]. The PoS mechanism is similar to the traditional PoS algorithm. Anyone can become a minter, known as *Investor* in Gridcoin terminology, by owning a certain amount of Gridcoin and participating in the minting process. In addition to this, other users, known as *Researchers* in Gridcoin terminology, can also participate in the minting process. Interestingly, an investor can also be a researcher and thus, can increase their amount of grid coin earned.

For this, a researcher installs the BOINC software and registers a project from the **BOINC** whitelist with his email address. The researcher is assigned a unique cross project identifier (CPID) and starts downloading the work share. Once the computation is completed, the researcher returns the result with a credit recommendation for the completed workload. The recommendation is compared with that of another researcher, and the minimum credit is rewarded. This workload credit data is stored in the header of each block and the researcher is rewarded with the corresponding amount of Gridcoin. To summarise, the consensus mechanism is mostly dominated by the PoS mechanism with Proof-of-BOINC acts as a reward mechanism for sharing unused computing resources available to the researchers. Hence, its security is similar to that of the traditional PoS algorithm.

2) Slimcoin's Proof-of-Burn (PoB). The Proof-of-Burn is a consensus algorithm proposed by Ian Stewart as an

Table 17: Comparing structural properties of PoS Consensus Algorithms.

Consensus /System	Node type	Single committee			Multiple committee		Mechanism
		Type	Formation	Configuration	Topology	Configuration	
Chained (PeerCoin)	Clients, Miners & Minters	-	-	-	Flat	Dynamic	Probabilistic lottery
Chained (CFFG)	Clients, Miners & Validators	-	-	-	Flat	Dynamic	Probabilistic lottery
BFT (Tendermint)	Clients & Validators	Open (Close)	Explicit	Dynamic (Static)	-	-	Voting
BFT (CTFG)	Clients & Validators	Open	Explicit	Dynamic	-	-	?
BTFG (Ouroboros)	Clients, Electors & Stakeholders	Open	Explicit	Dynamic			Voting
DPoS	Clients & Validators	Open	Explicit	Dynamic	-	-	Voting

Table 18: Comparing security properties of PoS Consensus Algorithms.

Consensus /System	Authentication	Non-repudiation	Censorship resistance	Attack Vectors		
				Adversary tolerance	Sybil protection	DoS Resistance
Chained (PeerCoin)	X	✓	High	$3f+1$	✓	✓
Chained (CFFG)	X	✓	High	$3f+1$	✓	✓
BFT (Tendermint)	✓ (In close type), X (In open type)	✓	High	$3f+1$	✓	✓
BFT (CTFG)	X	✓	High	$3f+1$	✓	✓
BFT (Ouroboros)	X	✓	High	$2f+1$	✓	✓
DPoS	X	✓	High	$3f+1$	✓	✓

Table 19: Comparison of additional attack vectors protection among PoS Consensus Algorithms

Consensus\System	Nothing-at-Stake	Bribing	Long-range	Coin-age	Pre-computing	Cartel formation
Chained (PeerCoin)	X	X	X	X	X	✓
Chained (Casper FFG)	✓	✓	✓	✓	✓	X
BFT (Tendermint)	✓	✓	✓	✓	✓	✓
BFT (CTFG)	✓	✓	✓	✓	✓	✓
BFT (Ouroboros)	✓	✓	✓	✓	✓	✓
DPoS	✓	✓	✓	✓	✓	X

Table 20: Comparing performance properties of PoS Consensus Algorithms.

Consensus\System	Fault tolerance	Throughput	Scalability	Latency	Energy consumption
Chained (PeerCoin, CFFG)	$2f+1$	Medium	Medium	Medium	Medium
BFT (Tendermint, CTFG, Ouroboros)	$2f+1$	High	High	Low	Low
DPoS	$3f+1$	High	High	Low	Low

alternative to PoW [163]. In PoW, miners need to invest in building a mining rig in order to participate in the mining process. In PoB, miners need to burn their coins in order to participate in the mining process. Burning coins mean that sending coins to an address without the private key and thus never usable. Thus, burning coins is an analogous idea to the investment for building a mining rig. The amount of burning has a positive correlation with the possibility of being selected for mining the next block. This is similar to the PoW system, where the miners increasingly invest in modern equipment to maintain the hash power, as the incentive decays with the complexity.

Slimcoin is a crypto-currency which utilises the idea of PoB in combination with PoW and PoS [164], [165], thus creating a hybrid consensus mechanism. Algorithmically, their idea is similar to the chained PoS algorithm of Peercoin as presented in in Section 5.2.1 with additional PoB mechanism sandwiched in between PoW and PoS algorithms. The PoW is used to generate the initial coin supply using the mechanism of Bitcoin. When the system has sufficient amount of money supply, it plans to switch to a hybrid of PoW and PoS mechanism similar to Peercoin where PoB will be used to select the miner. As this happens, the minters will need to burn their accumulated coins in order to be eligible to participate in the PoS minting process. Since PoB

algorithm is mostly used for minter selections, it has hardly any effect on the security of the system. Hence, its security and other properties are mostly similar to that of Peercoin.

3) Proof of Stake-Velocity (PoSV). One of the major limitations of coin-age based PoS is that there is no incentive (or lack of penalty thereof) for the minters to be online to participate in the staking process. This is because that the coin-age increases linearly over time, without the need for the stakeholders to be online and participate in the staking process. They can, therefore, choose to participate for a short period and then collect the reward and may go offline again. The lack of participants may facilitate attacks at a certain time.

To counteract this problem, a crypto-currency called Reddcoin proposed a novel hybrid algorithm called Proof of Stake-Velocity (PoSV) [166], [168]. The central to the PoSV is the idea of a mechanism called the *velocity of stakes* coupled with any traditional PoS algorithm. Conceptually, the velocity of stake mirrors the notion of the velocity of money, a terminology from Economics implying the frequency of money flow within the society [169]. Indeed, the velocity of stakes evolves around the idea of increasing the flow of stakes during the PoS consensus mechanism [167]. This (the flow of stakes) can be achieved if the minters are encouraged to actively participate in the consensus mechanism by staking their crypto-currency, instead of holding their coins offline. This process in a way will also increase the overall security of the system and counteract the lack of participant issue in PoS.

To facilitate this PoSV introduces a non-linear coin-ageing function in which the coin-age of a particular coin is gained much faster in the first few days and weeks than the gain in later weeks. For example, it has been estimated that minters who stake their coins every two weeks or less, can earn up to 20% more than people who do not participate in the staking process [167]. Such incentives encourage the minters to increase the velocity of stakes in the whole network. Note that PoSV is similar to any PoS mechanism along with its properties and hence, not explored in detail here.

6.2 N-POS/POW

The consensuses algorithms presented in this category do not rely any way on either PoW or PoS algorithms. Instead, they rely on completely novel mechanisms. Therefore, we call them N-POS/PoW algorithms for the convenience of group naming.

1) Proof-of-Cooperation (PoC). The Proof-of-Cooperation is a consensus algorithm introduced by the FairCoin crypto-currency [170], [171]. This consensus algorithm relies on several special nodes known as Certified Validating Nodes (CVNs). CVNs function similar to the way validators act in a DPoS consensus algorithm as utilised by EOS or Tron crypto-currencies, as they are nodes which can create blocks in Faircoin using the PoC consensus algorithm. However, unlike any DPoS validators, each CVN node is authenticated by their corresponding Faircoin identifier as well as trusted following a set of community-based rules and technical requirements [171]. The community rules state that a candidate node willing to be a CVN must participate in Faircoin community activities by performing some tasks. Examples of these tasks are running a local node or contributing to any technical or management issue related to Faircoin which must be confirmed by at least two active members of the community. Besides, the candidate node must follow a set of technical requirements such as 24/7 network availability and a special cryptographic hardware used for signature generation.

With the involvement of CVNs selected in the previously discussed manner, the core mechanism for PoC consensus algorithm is briefly discussed next. Blocks in Faircoin are created in a round-robin fashion in every three minutes of epoch by one of the CVNs. To create a new block, a CVN needs to be selected using a deterministic voting mechanism individually carried out by every single CVN in the network. The steps of this mechanism are:

- Each CVN finds the CVN, which has created a block furthest in the chain by traversing backwards through the chain.
- Next, it is checked if the found CVN has been active recently in the network by looking for its signature in the last few blocks. If so, this CVN will be selected as the next CVN.
- Then, each node creates a data set consisting of the hash of the last block, the ID of the selected CVN for the next block, and its own CVN ID, which is then signed by the specified cryptographic hardware. The created dataset, along with the signature, is then propagated through the network.
- The selected CVN receives this dataset along with their signature from multiple CVNs and verifies each signature. As soon as the selected CVN finds that more than 50% CVNs have selected it to be the next block creator, it can be certain that its turn is next at the end of the current epoch, i.e., three minutes.
- The selected CVN adds all pending transactions into a new block, along with all the received signatures, and propagates the block in the network.
- Upon receiving the block, other CVNs verify the block by checking the if the CVN who created the block is actually the one selected as the block creator as well as validating all signatures in it and its transactions. If the verification is successful, the block is added to the blockchain and the same mechanism continues.

2) Proof of Importance (PoI). PoS gives an unfair advantage to coin hoarders. The more coins they keep in their accounts, the more they earn. This means the rich get richer and everyone has an incentive to save coins instead of spending them. To solve these issues NEM has introduced a novel consensus mechanism called "Proof of Importance (PoI)" [172]. It functions similarly to PoS: nodes need to 'vest' an amount of currency to be eligible for creating blocks and are selected for creating a block roughly in proportion to some score. In Proof-of-stake, this 'score' is one's total vested amount, but in PoI, this score includes more variables. All the nodes that have more than 10000 XEM (the corresponding crypto-currency of XEM) are theoretically given equal positive importance and with 9B XEM coins there can be maximum $900K$ such nodes. However, the actual number of such nodes and their importance vary with time and their

amount of transaction in XEM.

The calculations borrow from the math of network clustering and page ranking. At a high level, the primary inputs are:

- Net transfers: how much has been spent in the past 30 days, with more recent transactions weighted more heavily.
- Vested amount of currency for purposes of creating blocks.
- Cluster nodes: accounts that are part of interlinked clusters of activity are weighted slightly more heavily than outliers or hubs (which link clusters but not part of them).

In NEM, the importance of an account depends only on the net transfers of XEMs from that account. To be considered for the importance estimation at a certain block height, h , a node must have transferred at least 100 XEMs during the last 30 days or $43,200$ blocks. The "importance score" addresses two primary criticisms of proof-of-stake.

One risk is that people hoard many coins as possible and reap the rewards from block creation. This concentrates wealth while discouraging transactions. The importance score means that hoarding will result in a lower score while spreading XEM around will increase it. Being a merchant pays better than having a hoard.

6.3 Analysis

In this section, we summarise the properties of different Hybrid and N-Pow/PoS algorithms utilising the taxonomies in Table 21, Table 22, Table 23 and Table 24. Like before, '-' signifies that the corresponding property is not applicable for the respective consensus algorithm, '?' indicates that the information the property has not been found, a '✓' is used to indicate an algorithm satisfies a particular property and 'X' is used to imply the reverse (not satisfied).

Table 21 presents the comparison of structural properties for the corresponding consensus algorithms. Among them, PoR and PoB depend on a multiple committee formation with a flat topology and dynamic configuration. Conversely, PoSV and PoI use an open single committee with a dynamic configuration, and probabilistic lottery as their underlying mechanism. PoC has an implicit, open, and dynamic single committee, which relies on voting mechanism.

All these algorithms have an adversary tolerance of $3f+1$ with the support of non-repudiation, Sybil protection, DoS resistance, and high censorship resistance as reported in Table 22. Entities in PoB, PoSV, and PoI do no require to be authenticated while PoC entities must be authenticated, and researchers in PoR need to be authenticated. However, other entities in PoW can remain non-authenticated, as indicated with the 'X' symbol in the table. All of them except PoC and PoI have $3f + 1$ adversary tolerance because of their usage of PoS algorithms. We have not found any regarding adversary tolerance for PoC and PoI.

Table 23 presents the comparison of some additional attack vectors for the Hybrid algorithms. As evident from the table, since these algorithms utilise PoS as one of their consensus algorithms, they suffer from the similar limitations of any PoS algorithm. For example, none of them has any guard against most of these additional attack vectors. The only exception is PoB which is because of its use of Peercoin like functionality, can resist the cartel formation attack.

The comparison of the performance properties for these algorithms is presented in Table 24. All of them have $2f + 1$ fault tolerance except PoC and PoI as we have not found any information fault tolerance for PoC and PoI. In terms of Scalability, Latency and Energy, every algorithm except PoB exhibits similar characteristics: they have high throughput, consume low energy, and have low latency, meaning they reach finality quickly. Because of its reliance on PoW, PoB has low scalability, low latency, and also consume meidum energy. In terms of throughput, PoR, PoSV and PoI have high throughput, whereas PoC has a low throughput and PoB has a medium throughput.

Finally, a comparison of the selected Hybrid and N-PoW/PoS crypto-currencies is presented in Table 25.

7 Non-incentivised Consensus

In this section, we present non-incentivised consensus algorithms that are used in private blockchain systems well-suited for non-crypto-currency applications. These algorithms are mostly based on classical consensus algorithms with special features added for their adoption for the corresponding blockchain systems.

One of the major initiatives within the private blockchain sphere is the Hyperledger project, which is an industry-wide effort [206]. Founded by the Linux Foundation, it is a consortium of some of the major tech vendors of the world. It provides an umbrella to facilitate the development of different types of open source projects utilising private blockchains with a specific focus to address issues involving business and governmental use-cases. Currently, there are six major projects within Hyperledger: Hyperledger Fabric [207], Hyperledger Sawtooth [208], Hyperledger Burrow [209], Hyperledger Iroha [210] and Hyperledger Indy [211]. Each of them is analysed below with a brief introduction.

7.1 Hyperledger Fabric

Hyperledger Fabric is the first major private blockchain system that originated from the Hyperledger ecosystem [207]. It has been designed with strong privacy in mind to ensure that different businesses organisations, including governmental entities, can take advantage of a blockchain system in different use-cases. A crucial capability of Fabric is that it can maintain multiple ledgers within its ecosystem. This is a useful feature, which separates Fabric from other blockchain systems consisting of only one ledger in each of their domains.

A key strength of Fabric is its modular design and pluggable features. For example, Fabric is not dependant on a particular format of ledger data, which is useful in several use-cases. In addition, the consensus mechanism is fully pluggable. Therefore, different types of consensus algorithms can be used in different situations.

As part of its consensus process, Fabric utilises a special entity called Orderer, which is responsible for creating a new block and extending the ledger by adding the block in the appropriate order. In addition, there are other entities known as endorsers. Each endorser is responsible for

Table 21: Comparing structural properties of Hybrid and N-POS/POW Consensus Algorithms.

Consensus /System	Node type	Single committee			Multiple committee		Mechanism
		Type	Formation	Configuration	Topology	Configuration	
PoR	Clients (Researchers) & Minters	-	-	-	Flat	Dynamic	Probabilistic lottery
PoB	Clients, Miners & Minters	-	-	-	Flat	Dynamic	Probabilistic lottery
PoSV	Clients & Minters	Open	Implicit	Dynamic			Probabilistic lottery
PoC	Clients & CVNs	Open	Explicit	Dynamic	-	-	Voting
PoI	Clients & transaction partners	Open	Implicit	Dynamic	-	-	Probabilistic lottery

Table 22: Comparing security properties of Hybrid and N-POS/POW Consensus Algorithms.

Consensus	Authentication	Non-repudiation	Censorship resistance	Attack Vectors		
				Adversary tolerance	Sybil protection	DoS Resistance
PoR	X/✓	✓	High	$3f+1$	✓	✓
PoB	X	✓	High	$3f+1$	✓	✓
PoSV	X	✓	High	$3f+1$	✓	✓
PoC	✓	✓	High	?	✓	✓
PoI	X	✓	High	?	✓	✓

Table 23: Comparison of additional attack vectors protection for Hybrid and N-POS/POW Consensus Algorithms

Consensus /System	Nothing-at-Stake	Bribing	Long-range	Coin-age	Pre-computing	Cartel formation
PoR	X	X	X	X	X	X
PoB	X	X	X	X	X	✓
PoSV	X	X	X	X	X	X

Table 24: Comparing performance properties of Consensus Algorithms of Hybrid and N-POS/POW.

Consensus	Fault tolerance	Throughput	Scalability	Latency	Energy
PoR	$2f+1$	High	Medium	Low	Low
PoB	$2f+1$	Medium	Low	Medium	Medium
PoSV	$2f+1$	High	Medium	Low	Low
PoC	?	LoW (10.6 TPS [173])	Medium	Low	LoW
PoI	?	High	Medium	Low	Low

Table 25: Hybrid & Non-PoW/PoS currencies

Currency	Genesis date (dd.mm.yyyy)	Block reward	Total supply	Consensus	Block Time
Gridcoin	24 Mar 2016	Minting	42 Million	PoR, PoS	1 minute
Slimcoin	May 2014	50-250 coins	133 Million	PoB, PoW, PoS	1.5 minutes
Reddcoin	January 20, 2014	Block reward	2.8 Billion	PoSV	1 minute
Faircoin	6th of March, 2014.	Block reward	5.3 Million	PoC	Depends on Time-weight Parameter
Burst	11 August 2014	Reduces at a fixed rate of 5 percent each month	204 Million	PoC	4 minutes
NEM	March 31st, 2015	transaction fees only + node rewards	899 Million	PoI	1 minute

validating and endorsing a transaction where it checks if an entity is allowed to perform a certain action in a ledger encoded within the transaction. Other participating entities are general users who create transactions. All the entities, including the Orderer(s) and the endorsers, are registered and authenticated via a Fabric specific special entity called *Membership Service Provider* (MSP). The MSP is responsible for managing the identities of all participants in the ledger. Using this identity layer, it is possible to create security policies that dictate which entities can perform what actions within a specific ledger. A simple flow of a consensus process in Fabric is illustrated in Figure 13.

The number of Orderer can be increased to distribute the ordering service. Currently, it supports SOLO and Kafka. A SOLO ordering service consists of just one single orderer and hence, cannot provide any type of fault tolerance. That

A All required entities are registered in the MSP.
B A channel with a ledger is initiated. In addition, a policy is created containing the endorsement criteria as well as other security and privacy criteria.
C A chaincode (smart-contract written either in Java or Go) is deployed in the ledger.
D When an entity wishes to invoke certain functions in the chaincode to read data from the ledger or to write data into the ledger, it submits a transaction proposal to all the required endorsers as dictated in the policy.
E Each endorser validates the proposal, executes the chaincode and returns a proposal response consisting of other ledger data.
F The proposal, its response, and other ledger data are encoded as a transaction and sent to the Orderer.
G The Orderer creates a block using the transaction and returns the block to the endorsers.
H Each endorser validates the block and, if validated, extends the ledger by attaching the new block. This essentially updates the state of the ledger.

Figure 13: A simple flow of a consensus process in Fabric.

is why it is not recommended to utilise the SOLO model in the deployed system and has only been provided for initial testing. On the other hand, the Kafka Orderer utilises a Kafka cluster for deploying distributed Orderers. Kafka is a distributed streaming platform with a pub-sub architecture [212] and is coupled with Zookeeper, a distributed coordination service [213]. At this point, the Kafka Orderer is the only recommended setting for achieving consensus in Fabric. An SBFT (Simplified Byzantine Fault Tolerance) based consensus algorithm is currently being developed and is to be released soon.

7.2 Hyperledger Sawtooth

Hyperledger Sawtooth, initially developed by Intel, is a software framework for creating distributed ledgers suitable for a variety of use cases [208]. Sawtooth utilises a novel consensus algorithm called Proof-of-Elapsed-Time (PoET), which depends on Intel SGX (Software Guard Extension). Intel SGX is a new type of Trusted Execution Environment (TEE) integrated into the new generation of Intel processors. SGX enables the execution of code within a secure enclave inside the processor, whose validity can be verified using a remote attestation process supported by the SGX.

PoET, similar to the Nakamoto consensus algorithm in Bitcoin, relies on the concept of electing a leader in each round to propose a block to be added in the ledger. The difference is that the Nakamoto algorithm and its variants select a leader by a lottery mechanism, which utilises computing power to generate a proof, as described previously. However, PoET solely relies on the Intel SGX capability to elect a leader. During each round, every validator node in the network, requests for a wait time from a trusted function in the SGX enclave. The validator that is assigned the shortest waiting time is elected as the leader for that round. The winning validator then can propose a block, consisting of a series of transactions from the defined transaction family. Other validators can utilise a trusted function supported by SGX to assess whether a trusted function has assigned the shortest time to the winning validator, and the winning validator has waited the specified amount of time. Furthermore, other validators verify the validity of the block before it is included in the ledger. The inclusion of the PoET as a consensus algorithm enables Sawtooth to achieve massive scalability as it does not need to solve a hard, computationally intensive cryptographic puzzle. In addition, it allows Sawtooth to be used not only for a permissioned ledger, but also for a public ledger.

7.3 Hyperledger Burrow

Hyperledger Burrow is a private (permissioned) deployment of the Ethereum platform [209]. It has been created and then deposited to the Hyperledger code-base by Monax Industries Limited [214]. The core component in Burrow is a permissioned version of the EVM (Ethereum Virtual Machine) to ensure that only authorised entities can execute code. Two additional components have been added: Byzantine fault-tolerant Tendermint protocol [179], [221] and the RPC gateway.

The Tendermint consensus falls under the category of a Byzantine Fault Tolerance (BFT) algorithm, which can be used to achieve consensus even under the Byzantine behaviour of a certain number of nodes as presented in Section 5.2.2.

Burrow depends on several validators, which are known (authorised) entities with the duty to validate each block utilising the Tendermint consensus algorithm. This algorithm allows consensus to be achieved in Burrow with $1/3$ nodes exhibiting Byzantine behaviour, either acting maliciously or having been down due to network or system failure.

Since Burrow utilises the EVM, a wide-range of smart-contracts and DApps (Decentralised Applications) could be deployed. Using the Tendermint algorithm with a set of known validators allows Burrow to scale at a much faster rate than Ethereum while preserving the privacy of transactions by allowing only known entities to participate in the network.

7.4 Hyperledger Iroha

Hyperledger Iroha is a private blockchain system initially developed by Soramitsu, Hitachi, NTT Data, and Colu and is currently hosted by Linux foundation under the Hyperledger Project [210], [215]. Iroha aims to create a simple blockchain infrastructure which can be incorporated into any system which requires a blockchain architecture underneath to function. The major emphasis while designing Iroha is on a simpler construction with a strong focus on mobile-friendly application development using a novel consensus mechanism called YAC (Yet Another Consensus) [215], [216]. One fundamental different of Iroha from other Hyperledger project is its fine-grained permission control mechanism which allows defining permissions for all relevant commands, queries, and even joining in the network.

The core architecture consists of several components [216], [219]. A brief description of its major components is presented below:

- **Troii** represents the entry point of any application to the Iroha network. It utilises gRPC (gRPC Remote Procedure Calls [218]), an open source RPC framework, to interact with different peers and entities within the blockchain network.
- **Model** represents how different entities are represented within the system and defines the mechanism to interact with them.
- **Network** provides the network functionalities required to maintain the P2P network and to propagate transactions in the network.
- **Consensus** facilitates the functionalities related to achieving consensus in the network using the YAC consensus protocol, a practical byzantine fault-tolerant algorithm (discussed below).
- **Simulator** provides a mechanism to simulate the effects of transactions on the chain by creating a temporary snapshot of the chain state.
- **Validator** allows the validation of transactions by verifying its formats and signature along with the verification of business rules and policies involved in the transactions. There are two types of validations in Iroha:
  - *Stateless* validation checks for transaction formats and signature.
  - *Stateful* validation checks the business rules and policies, e.g., if a certain action is allowed by an entity.
- **Synchroniser** is a part of the consensus component and is responsible for synchronising the chain to a new or disconnected node.
- **Ametsuchi** is the storage component of Iroha and is used to store the blocks and the chain state known as World State View (WSV).

These components are used by three core entities within the architecture [216]:

- Clients are applications that they can query data from the allowed Iroha chain as well as can perform certain actions, called commands, by which the state of the chain is updated. For each of these, clients need to interact with the peer.
- Peers are nodes that have the following two functionalities:
  - To maintain a copy of the ledger. Applications can thus interact with a peer to query a chain or to submit transactions to update the chain.
  - To participate in the consensus process by maintaining its address, identity and trust as a single entity in the network.
- Ordering service node(s): Like Fabric, ordering service nodes are responsible for ordering transactions and creating a proposal of a block.

With these components and entities, a flow of transactions in *Iroha* is briefly presented in Figure 14 [216].

A A client prepares and sends a transaction to a peer using Troii.
B The peer performs stateless validation to the transaction and forwards the transaction to the ordering service using an ordering gate.
C The ordering service combines and orders transactions from different peers in a transaction proposal which is then broadcast to the peers.
D Each peer performs a stateful validation of the proposal using the simulator and creates a block consisting of only verified transactions. Each peer signs the block, generates a hash of the proposed block and finally, creates a tuple containing the hash and the signature. Such a tuple is called a *vote*. The block and the vote are then internally sent to the consensus gate to initiate the YAC mechanism.
E The YAC mechanism in each peer prepares an ordered list of voting peers utilising the hashes created in the previous step. The first peer in the list is regarded as the *leader* and is responsible for aggregating votes from other voting peers.
F After aggregating all votes from the voting peers, the leader computes the supermajority (usually 2/3rd) of votes for a certain hash (signifying a block).
G Once a supermajority for a proposed block is achieved, the leader propagates a commit message for this particular block to all voting peers.
H Each voting peer verifies the commit message and adds the block to the blockchain.

Figure 14: A flow of transactions in *Iroha*.

7.5 Hyperledger Indy

Hyperledger Indy is a private blockchain system purposefully built for providing an ecosystem for blockchain-based self-sovereign identity [211], [222]. The concept of Self-Sovereign Identity has been initially promoted by the Sovrin foundation [223], a non-profit international entity consisting of several private organisations to promote the notion of Self-sovereign Identity. The Indy project is closely associated with the Sovrin foundation focusing on materialising this notion of a self-sovereign identity system as a public identity utility.

Currently, Indy consists of the following two major components:

- **Indy-plenum**: Plenum is the underlying distributed ledger (blockchain) construct of the Indy platform. Like any distributed ledger, the Plenum ledger is fundamentally an ordered log of transactions. In addition, it consists of several nodes, among which a single or a few chosen ones act as the leader responsible for ordering the transactions. The nodes execute a consensus protocol which utilises a three-phase commit to reach agreement among themselves regarding the order of the transactions.
- **Indy-SDK**: This provides the required software APIs and tools to enable other software to interact with the Plenum ledger. It hides all the intricate internals from the users of the platform so that the platform can be utilised without even knowing the complexities of the ledger and its associated consensus protocol.

The consensus protocol utilised in Indy is called RBFT (Redundant Byzantine Fault Tolerance) [224]. Like any other byzantine fault tolerance protocol, it relies on $3f + 1$ nodes (a participant in the consensus protocol) in order to handle f byzantine nodes [224], [225]. For example, it requires

four deployed nodes in order to handle a single byzantine node. Each participating node in RBFT deploys two (or more) protocol instances, aptly called Master and Backup protocol instance, each of which is executed in parallel. A separate primary node (also called a leader) is selected from the master and the backup protocol instance. The leader is responsible for ordering the transactions. Its performances, i.e., latency and throughput, are periodically observed by the other instances. If its performance degrades, a different leader is selected from the backup instance.

Indy maintains a number of ledgers for different purposes, unlike many other blockchain systems which employ a solo ledger. For example, separate ledgers are maintained for node maintenance, for identity transactions and so on. Clients (users via their appropriate software interfaces) can interact with these ledgers via different nodes for updating the ledger via transactions and for reading from the ledger via queries. A fine-grained permission mechanism can be used to dictate which client has to write permissions, however, any client can read from the ledger.

Once a node receives a transaction from a client, it performs some validation and then broadcasts the transaction to other nodes in the network. When the transaction reaches enough nodes, the primary node starts a new consensus round using a three-phase commit mechanism. In the end, all nodes agree to the order proposed by the primary node and add the transaction into the corresponding ledger.

7.6 Analysis

In this section, we analyse the non-incentive consensus protocols against the criteria selected before. Block and reward properties are not considered as they are not relevant for non-incentivised consensus protocols. We use the notation '✓' to indicate an algorithm satisfies a particular property and the notation '-' to indicate that there is no information regarding that specific property. For other properties, explanatory texts are added.

Structural properties. Tn Table 26, we present the comparison of structural properties among the non-incentivised consensus algorithms discussed in this section. As evident from the table, different algorithms use different types of nodes, and all algorithms are based on single committee with closed committee type and explicit committee formation. Only YAC relies on a dynamic configuration which utilises the reputations of the nodes from previous interactions; all others have the static configuration.

Voting is the predominantly used underlying mechanism, which is utilised by Tendermint Burrow, YAC, and RBFT, whereas PoET relies on a lottery mechanism. Fabric currently utilises the ordering services by the orderer. In the future, it might utilise SBFT, which leverages the voting mechanism.

Security properties. The comparison of security properties among the non-incentivised consensus algorithms is presented in Table 27. All algorithms support non-repudiation via digital signature and have a significantly low censorship resistance. This is because the identities of all participating nodes are known. In case any node starts misbehaving, because of an attacker taking control of that node, it can be easily identified, and proper actions can be taken. The same logic applies for the Sybil protection and towards DoS resistance. Being mostly based BFT algorithms, all algorithms, except PoET, have $3f + 1$ adversarial tolerance. It has been found that PoET has an adversarial tolerance of $\Theta\left(\frac{log\ log\ n}{log\ n}\right)$ [220], where n is the number of nodes.

Performance properties. The comparison of performance properties among the non-incentivised consensus algorithms is presented in Table 28. All algorithms can provide a good throughput and do not require to consume any significant amount of energy. PoET, utilising a lottery mechanism, can be scaled with a large number of validators, however, this will increase the latency (finality) of transactions [217]. All other algorithms employing a voting mechanism cannot be scaled with a large number of validators, providing low latency for the transactions. Fabric, YAC, and RBFT provide a $2f + 1$ fault tolerance, whereas the information regarding the fault tolerance for PoET and Tendermint Burrow is not specified formally.

8 Discussion

As per our analysis in different sections, it is clear that PoW consensus algorithms have major limitations, specifically in terms of power consumption and scalability. Many regard PoS, and it is variant DPoS, to be the most suitable alternatives. To understand the applications of these algorithms in public blockchain systems, we have analysed the top 100 crypto-currencies, as reported on CoinMarketCap [4] as of 18 July, 2019.

In the first analysis, we have calculated the number of consensus algorithms used in these (top 100) crypto-currencies. The distribution of consensus algorithms is presented in Figure 15 . As per our analysis, PoW is still the most widely used (57%) consensus algorithms to date, whereas DPoS is the second most with 11%, and PoS is the third most with 6% used consensus algorithms. All other consensus algorithms represent the remaining 26%. This means that, even though many consider that PoS and DPoS are the best alternatives to PoW, their adoption is still far behind PoW.

To investigate it further, we have analysed a year-wise distribution of the genesis dates of different crypto-currencies. It is to understand if there is any inclination towards an alternative consensus algorithm over PoW in recent years. The distribution is illustrated in Figure 16, which represents a surprising observation: PoW is still the most widely used algorithms for crypto-currencies which have been created in recent years. For example, the numbers of crypto-currencies created with PoW algorithms in last three years (2017, 2018 & 2019) are 11, 19 and 4 respectively, in comparison to 4, 2 and 2 for PoS and DPoS combinedly. This implies that PoW is still the most popular consensus algorithm among the crypto-currency community. A deeper investigation reveals another insight though. The top 100 list retrieved from Coinmarketcap also contains crypto-tokens generated on top of any smart-contract platform such as Ethereum, EOS and Tron with majority tokens are built on

4. https://coinmarketcap.com/

Table 26: Comparing structural properties of Consensus Algorithms of Hyperledger Systems.

Consensus /System	Mechanism	Node type	Single committee		
			Type	Formation	Configuration
Fabric	Ordering/Voting (SBFT)	Client (Regular peer), Endorsing peer & Orderer	Close	Explicit	Static
PoET	Lottery	Client, Transaction processor & Validator	Close	Explicit	Static
Tendermint Burrow	Voting	As Tendermint (3)	Close	Explicit	Static
YAC	Voting	Client, Peer & Ordering Service node	Close	Explicit	Dynamic
RBFT	Voting	Client & Node	Close	Explicit	Static

Table 27: Comparing security properties of Consensus Algorithms of Hyperledger Systems.

Consensus	Non-repudiation	Censorship resistance	Attack Vectors		
			Adversary tolerance	Sybil protection	DoS Resistance
Fabric	✓	Low	$3f+1$	✓	✓
PoET	✓	Low	$\Theta\left(\frac{log\, log\, n}{log\ n}\right)$	✓	✓
Tendermint Burrow	✓	Low	$3f+1$	✓	✓
YAC	✓	Low	$3f+1$	✓	✓
RBFT	✓	Low	$3f+1$	✓	✓

Table 28: Comparing performance properties of Consensus Algorithms of Hyperledger Systems.

Consensus	Fault tolerance	Throughput	Scalability	Latency	Energy
Fabric	$2f+1$	Good	Medium	Low	Low
PoET	-	Good	Good	Medium	Low
Tendermint Burrow	-	Good	Medium	Low	Low
YAC	$2f+1$	Good	Medium	Low	Low
RBFT	$2f+1$	Good	Medium	Low	Low

top of Ethereum. Most of these tokens have emerged after 2016 with Ethereum utilising PoW . This could be the reason why the most recent crypto-currencies have been found to utilise PoW.

Another indication of PoW domination over other algorithms is the market-cap distribution of their corresponding crypto-currencies. The distribution is presented in Table 29 and illustrated in Figure 17. Not surprisingly, PoW currencies with a market-cap of around 221 Billion USD have a massive 93% dominance over other currencies. DPoS and PoS currencies are the nearest rivals with a market-cap of around 6 Billion USD and dominance of only 3% for each group.

Table 29: Market capitalisation of major consensus algorithms in top 100 Crypto-currencies

Consensus Algorithms	Market-cap (USD)
PoW	$221,238,526,412$
DPoS	$6,483,606,020$
PoS	$6,287,224,485$
PoW+PoS	$2,436,683,929$
Proof of Authority	$572,188,935$
Proof of Activity	$274,066,240$

From our investigation, it is clearly evident that PoW algorithm, even with its major limitations, is still the most popular consensus algorithm to be utilised in different crypto-currencies. Currencies which utilise PoW algorithms consume a significant amount of energy as illustrated in Section 5.1.4. Besides, they have a reduced throughput (in terms of transaction number) compared to PoS and DPoS currencies. For example, the reported TPS (Transactions Per Second) for Bitcoin and Ethereum are 7 and $15-25$, respectively [226], while DPoS currencies EOS has a reported and estimated TPS of 50 and 4000 respectively [226] and Tron has a claimed TPS of 2000 [227]. Clearly, DPoS currencies have better performance, at least in terms of TPS, over any PoW currency. Therefore, one might ask the underlying reason behind this counter-intuitive trend of PoW being the most popular consensus algorithm. We have identified a few reasons behind this as presented below:

- Bitcoin is the most dominant crypto-currency in terms of market-cap. As of 18 July, it has a market cap of around 171 Billion USD. In addition to this, its different forked variants (Bitcoin Cash [5] and Bitcoin Satoshi Vision [6]) also have a combined market-cap of 8 Billion USD. If we exclude Bitcoin and its variants, we have a slightly different distribution of market-cap, as illustrated in Figure 18. Here, the market-cap percentage of PoW algorithm is reduced from 93% to 71% percent, which is still significant in comparison to DPoS and PoS, its nearest rivals.
- PoW has the first-mover advantage because of Bitcoin and Ethereum, both being the pioneer in their respective domain. Bitcoin has been the first successful crypto-currency, while Ethereum is the first blockchain-based smart-contract platform. Other crypto-currencies, being motivated by their success, might have adopted the approach of utilising PoW as their corresponding consensus algorithm.

5. https://www.bitcoincash.org/
6. https://bitcoinsv.io/

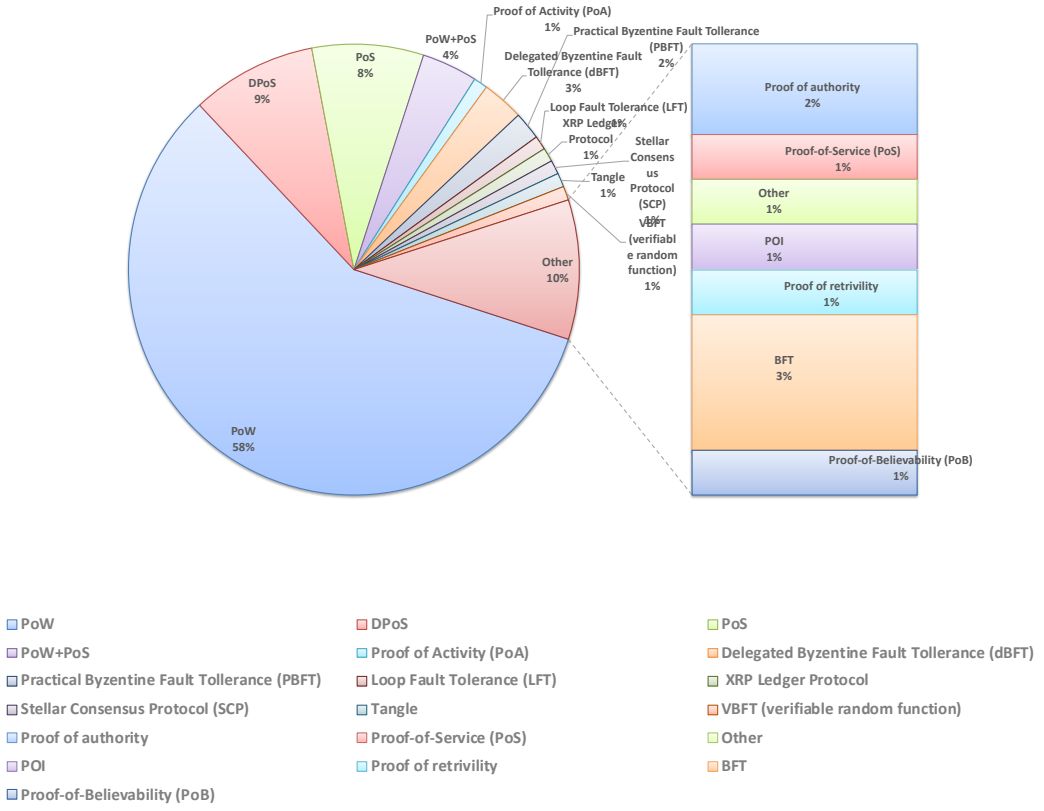


Figure 15: Consensus algorithms in Top 100 Crypto-currencies

- Another strong argument in favour of PoW is its underlying security. The number of miners is far greater in Bitcoin than the number of validators in PoS and DPoS. This implies a better decentralisation in Bitcoin than PoS or DPoS. For example, EOS has only 21 validators, while Tron has 27 validators. The probability of collusion among these validators is far greater than that of any popular PoW currency. For this reason, many in the blockchain community have been doubtful of the security of any PoS/DPoS currency. However, there is a counter argument against this. Because of the mining centralisation issue (highlighted in Section 5.1.4), many point out that a PoW algorithm might also be prone to centralisation. Therefore, a PoW currency might also suffer from collusion attack.

With the dominance of PoW over other consensus algorithms, one might wonder what lies ahead and might ask if there will be any shift of balance among the consensus algorithms. We believe that we will most definitely experiment with a shifting of balance in the near future. In this regard, the PoS transformation process of Ethereum will be a crucial factor. The proposed Ethereum PoS consensus mechanisms, both CFFG and CTFG, are highly regarded by the academics and industrial enthusiasts for their strong guarantee of security. With their strong focus on economic incentive and game-theoretic based approach, it is believed that their security will be as close as PoW and much better than any current PoS/DPoS algorithm can provide. In particular, the number of validators will be much higher than any number leveraged in the current PoS/DPoS algorithms. However, it is yet to be seen how they will perform once deployed in real-life settings.

The existence of numerous algorithms and wide variations in their properties impose a major challenge to comprehend them properly. In particular, it is often difficult to test the suitability of a particular algorithm under certain criteria. A visual tool would be a great help in this regard. Towards this aim, we present a decision tree in Figure 19, which can be used to determine the suitable consensus algorithms under certain criteria in different scenarios. For example, such a decision tree diagram can be leveraged to select a particular consensus algorithm while designing/developing a new blockchain system.

The tree utilises five critical criteria to achieve its goal: incentives, energy consumption, scalability, security (with respect to adversary tolerance), and ASIC-resistance. If the system needs to incentivise the miner/validating nodes, then proof-of-work(PoW) and proof-of-stake (PoS) consensus are appropriate choices. Because of their underlying incentives mechanisms, the primary applications of these consensus algorithms are public crypto-currencies. On the other hand, a private blockchain network usually does not rely on any crypto-currencies to motivate or incentivise any validators to run the blockchain network. In addition to incentives, energy consumption is another determining factor in choosing appropriate consensus algorithms. PoW-type algorithms consume high energy, whereas PoS algorithms and their derivatives consume a moderate amount of energy. PoW-types algorithms are very slow as of now and can process only a limited number of transactions.

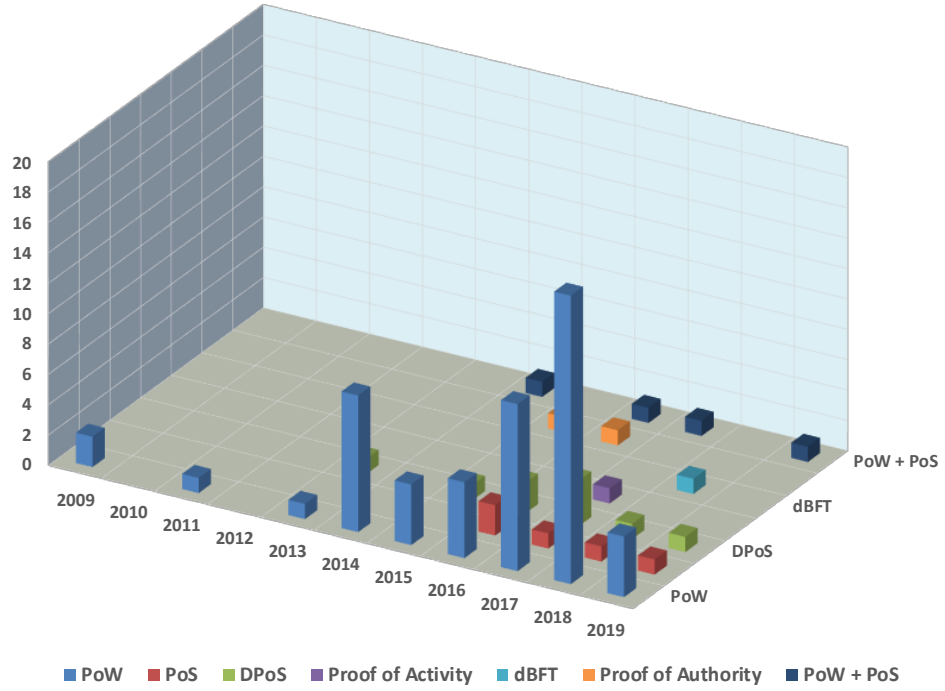


Figure 16: Year-wise distribution of consensus algorithms in Top 100 Crypto-currencies

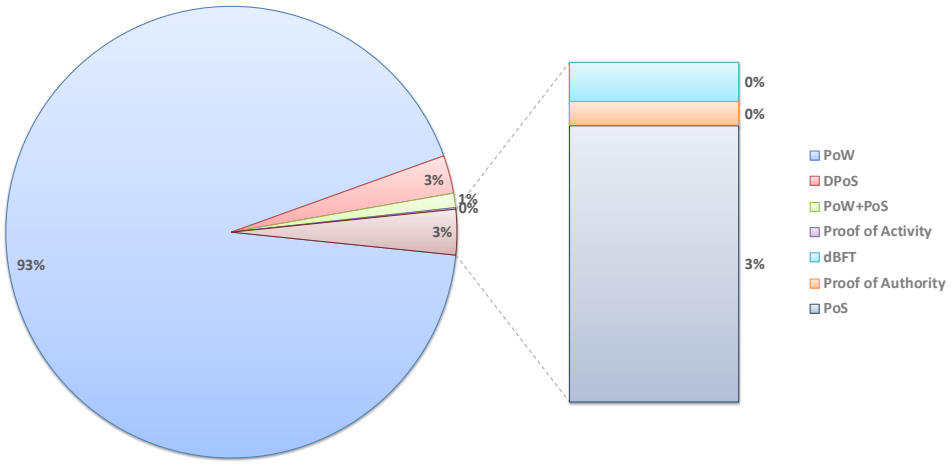


Figure 17: Percentage of market capitalisation of consensus algorithms in top 100 Crypto-currencies

However, compromising a popular PoW-based blockchain network is very difficult, and therefore, they are more secure than their counterparts. PoW-based algorithms can also be classified based on computational complexity. As discussed earlier, ASIC is a specialised hardware, designed and used to solve hash-based computational problems. ASIC is expensive and hinders common people from participating in the blockchain network. Therefore, memory-based PoW has been designed. Now it is widely used in different crypto-currencies. Non-incentivised consensus algorithms are mostly used in private blockchain systems. They consume a very low amount of energy compared to other types of consensus algorithms and are also very scalable. That means the miners can verify the transactions and create blocks really fast. However, a comparatively low number of validating nodes makes these algorithms more vulnerable to attacks.

For clarity, we provide a few examples to utilise the decision tree diagram presented in Figure 19. If an incentivised algorithm is required for a highly scalable blockchain system that aims to consume low energy DPoS and BFT derivatives such as Tendermint, CTFG, and Ouroboros are the preferred options. However, they will have moderate security as described earlier. On the other hand, if security is of the highest priority, PoW algorithms are more suitable. In this scenario, there are two options: memory-bound or CPU bound. If ASIC resistance is desired, one should opt for memory-bound PoW algorithms. However, in such a case, one has to sacrifice scalability, and such algorithms will consume high energy.

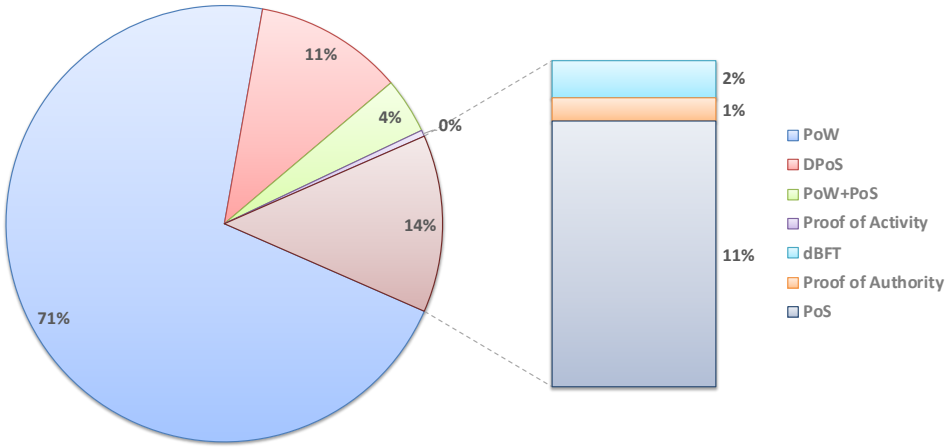


Figure 18: Percentage of market capitalisation excluding Bitcoin and its variants

Note that this is just an example of how such a diagram can be developed using our selected four criteria. Other criteria can be utilised to generate a different diagram which might be suitable for other specific scenarios. Whenever such a diagram is to be developed, the tables (Table 13, Table 14, Table 15, Table 17, Table 18, Table 19, Table 20, Table 21, Table 22, Table 23, Table 24, Table 26, Table 27 and Table 28) utilised to compare different consensus algorithms against the defined properties in the taxonomy will be crucial as the these tables will provide the required template by which such a diagram can be created.

9 Conclusion

With the popularisation of crypto-currencies, and blockchain in general, there has been a renewed interest in the practical implications of different distributed consensus algorithms. Most of the existing systems struggle to properly satisfy the need for any wide-scale real-life deployment as they have serious limitations. Many of these limitations are due to the underlying consensus algorithm used in a particular system. Therefore, in the quest to create more suitable practical blockchain systems, the principal focus has been on distributed consensus. This has led to the explorations; either existing consensus algorithms have been exploited or novel consensus mechanisms have been introduced. The ultimate consequence of this phenomenon is a wide-range of consensus algorithms currently in existence. To advance the knowledge of this domain, it is essential to synthesise these consensus algorithms under a systematic study, which is the main motivation of this article.

Even though there have been several similar works, this is the first paper to introduce a taxonomy of properties desirable for a consensus algorithm and then utilise that taxonomy to analyse each algorithm in a detailed fashion. In addition, different consensus algorithms have been grouped into two major categories: Incentivised and Non-incentivised consensus algorithms. An incentivised consensus algorithm, exclusively utilised by public blockchain systems and crypto-currencies, relies on incentives for the participants in order to motivate them to behave as intended. On the other hand, in any non-incentivised algorithm, the participants are considered as trusted, and hence, it is assumed that no incentives are required to ensure intended behaviour. As such, these algorithms are mostly used in the private blockchain sphere. We have again grouped incentivised algorithms into three major sub-categories: PoW (Proof of Work), Proof of Stake (PoS) and consensus algorithms beyond PoW and PoS.

A PoW algorithm relies on computational complexities or memory size/performance to solve a cryptographic puzzle. There are three major approaches followed by PoW consensus algorithms: i) a compute-bound PoW leveraging the capabilities of the processing unit, ii) a memory-bound PoW which is more reliant on the size and performance of the main memory, and iii) a chained PoW utilises a number of hashing algorithms executed consecutively one after another. Blockchain systems utilising such a mechanism has special nodes, called miner nodes, who are responsible for solving this puzzle and creating a new and valid block and extending the chain by appending this block in the existing chain. The probability to solve this puzzle depends on a network parameter, called difficulty, which is adjusted automatically after a certain period of time. As more miners participate in the network, the network parameters are adjusted in such a manner that requires more computational power to mine a new block. As the corresponding systems become more popular, it attracts more miners, which increases the security of the system. However, the increased computational power results in more energy being consumed. Apart from this, PoW systems generally have a low throughput and do not scale properly. PoS algorithms and their corresponding mechanisms have been analysed in greater detail in Section 5.1.

To alleviate the major issues of PoW, Proof of Stake (PoS) has been proposed. In PoS, the nodes who would like to participate in the block creating process are called minters, and they need to own and lock a certain amount of the corresponding crypto-currency, called stake. Such a stake is used to ensure that the minters will act as required since they will lose their stakes when acting maliciously. PoS has several variants: Chained PoS, BFT PoS and DPoS. The core idea of a chained PoS is to leverage a combination of PoW and PoS algorithms chained together to achieve consensus.

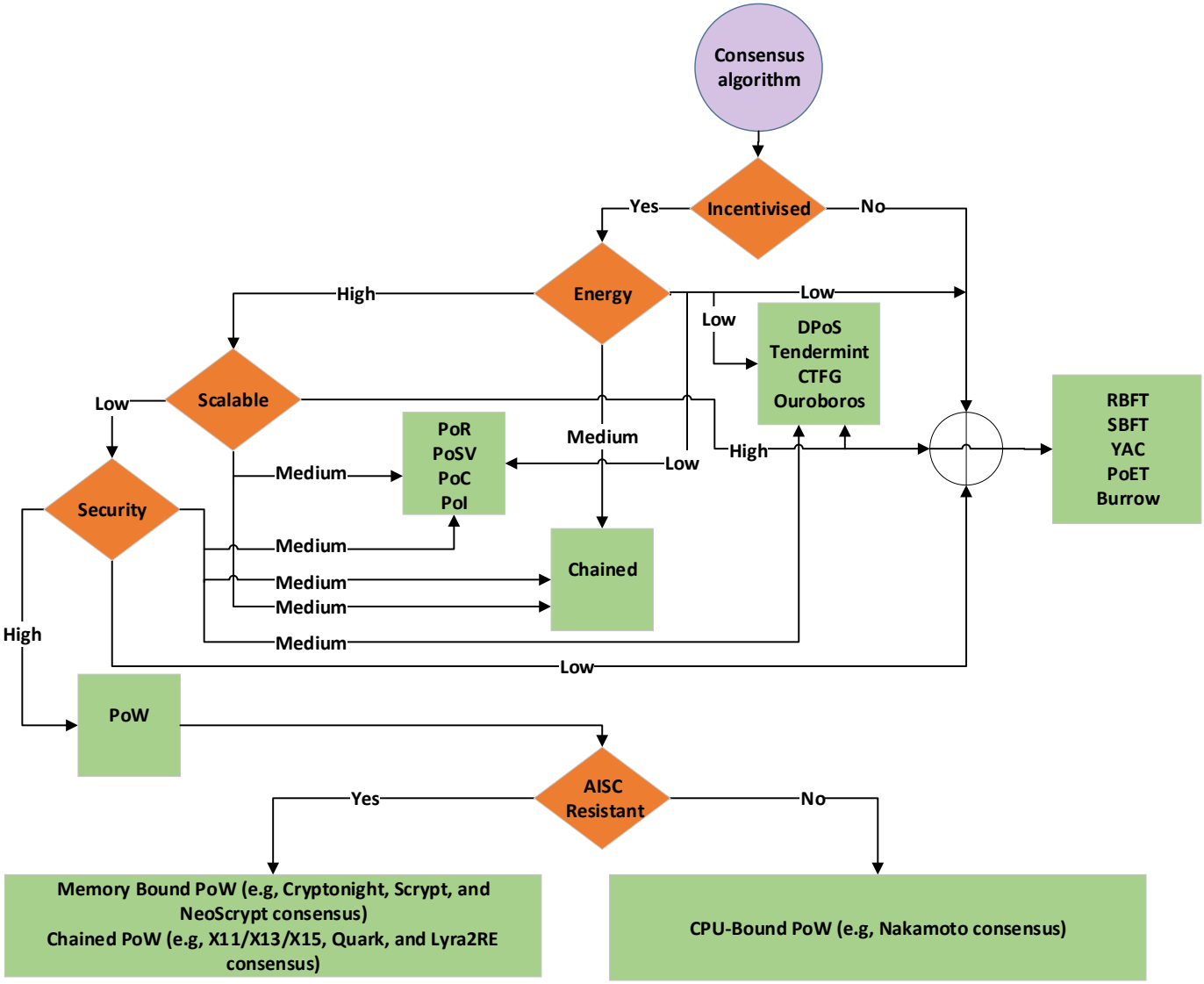


Figure 19: Decision tree to choose appropriate consensus algorithms

BFT PoS uses a multi-round PoS mechanism in which a validator (minter) is selected, from a set of validators, by the agreement of super-majority quorum among other validators. On the other hand, DPoS selects a minter, from a set of minters, using votes from other clients of the network. PoS algorithms are generally fast and scalable, having high throughput. However, they also need to consider several other attack vectors such as Nothing-at-stake, bribing, long-range attack, cartel formation, and so on. Detailed analysis of different aspects of PoS algorithms has been presented in Section 5.2.

There are also some Hybrid consensus algorithms that combine the mechanisms of PoW and/pr PoS with another novel algorithm. Proof of Research, Proof of Burn, Proof of Stake-Velocity are examples of such an algorithm. Again, there are mechanisms that are novel and have no reliance on PoW/PoS whatsoever. Proof of Cooperation and Proof of Importance are examples of such novel algorithms. The discussion and analysis of these consensus algorithms have been presented in Section 6.

Finally, there are also a few non-incentivised consensus algorithms which are exclusively utilised in private blockchain systems. Hyperledger is the leading private blockchain foundation under which different private blockchain systems such as Hyperledger Fabric, Hyperledger Sawtooth, Hyperledger Burrow, Hyperledger Iroha, Hyperledger Indy, and so on. These systems rely on different other consensus mechanisms such as SBFT, PoET, Tendermint Burrow, YAC, and RBFT. Key characteristics of these consensus algorithms are high throughput and low latency with acceptable scalability. Also, the algorithms require that every entity that participates in the network must be properly authenticated. A detailed analysis of these algorithms has been presented in Section 7.

Our analysis in Section 8 suggests that PoW, with its many disadvantages, still is the most dominant in terms of market capitalisation (indicating its adoption) and cryptocurrency in the world. As discussed earlier, DPoS and PoS algorithms, PoW's closest rivals, aim to tackle many of PoW's limitations. However, their adoption is still limited. In addition to this analysis, we have presented an exemplary

decision tree-based figure which can be utilised to filter out or select consensus algorithms that fit certain criteria. Such a figure will be a useful tool for any who would like to test the suitability of a certain consensus algorithm under certain criteria.

There is one issue that must be highlighted before we conclude this article. The principal focus of this article has been to explore and synthesise the consensus algorithms available in different blockchain systems. However, there are other distributed ledger systems, which do not rely on any blockchain-type structure. Instead, they utilise other structures to represent their respective ledgers. Examples of two such prominent crypto-currencies are IoTA [7] and NANO [8]. Both of their ledgers are based on DAG (Directed Acyclic Graph), a specific type of directed graph with no cycle. However, IoTA uses a novel consensus algorithm called Tangle [229] while NANO utilises a representative based consensus mechanism [228]. These two systems have received significant attention because of their fee-less structure and fast transaction rates. However, we do not consider these systems any further as they are out of scope for this article. We plan to investigate such novel systems in the future in a different exploration.

There is high anticipation among the blockchain enthusiasts that blockchain technology will disrupt many existing application domains. However, to unlock its true potential, a blockchain system must adopt a suitable consensus that can enable it to satisfy its intended properties. This is because a consensus algorithm is the core component of any blockchain system, and it dictates how a system behaves and the performance it can achieve. However, as our analysis in this article suggests, an ideal consensus algorithm is still elusive as almost all algorithms have significant disadvantages in one way or another with respect to their security and performance. Until a consensus algorithm finds the correct balance between these crucial factors, we might not see the wide-scale adoption as many crypto-currency enthusiasts are hoping.

REFERENCES

[1] Nakamoto, Satoshi "Bitcoin: A peer-to-peer electronic cash system". 2008. [online] Available: https://bitcoin.org/bitcoin.pdf.

[2] Pilkington, Marc "11 Blockchain technology: principles and applications". Research handbook on digital transformations, 225, 2016.

[3] Crosby, Michael and Pattanayak, Pradan and Verma, Sanjeev and Kalyanaraman, Vignesh and others "Blockchain technology: Beyond bitcoin". Applied Innovation, 2(6-10), p. 71, 2016.

[4] Cachin, C., and Vukoli, M. "Blockchains Consensus Protocols in the Wild". arXiv preprint arXiv:1707.01873, 2017.

[5] Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., and Danezis, G. "Consensus in the Age of Blockchains.". arXiv preprint arXiv:1711.03936, 2017.

[6] Wang, W., Hoang, D.T., Hu, P., Xiong, Z., Niyato, D., Wang, P., Wen, Y. and Kim, D.I. "A survey on consensus mechanisms and mining strategy management in blockchain networks". IEEE Access, 7, pp.22328-22370, 2019.

[7] Baliga, A. "Understanding Blockchain Consensus Models". April, 2017. [Online] Available: https://www.persistent.com/wp-content/uploads/2017/04/WP-Understanding-Blockchain-Consensus-Models.pdf.

7. https://www.iota.org/
8. https://nano.org/en

[8] Sankar, Lakshmi Siva and Sindhu, M and Sethumadhavan, M "Survey of consensus protocols on blockchain applications" 4th International Conference on Advanced Computing and Communication Systems (ICACCS). IEEE, 1–5, 2017.

[9] Seibold, Sigrid and Samman, George "Consensus: Immutable agreement for the Internet of value". https://assets.kpmg.com/content/dam/kpmg/pdf/2016/06/kpmgblockchain-consensus-mechanism.pdf KPMG. 2016.

[10] Mukhopadhyay, Ujan and Skjellum, Anthony and Hambolu, Oluwakemi and Oakley, Jon and Yu, Lu and Brooks, Richard. "A brief survey of cryptocurrency systems." In Proceedings of the 14th annual conference on privacy, security and trust (PST). IEEE, 745–752, 2016.

[11] Schneider, F. B. "Implementing fault-tolerant services using the state machine approach: A tutorial". ACM Computing Surveys (CSUR), 22(4), 299-319, 1990.

[12] C. Cachin, R. Guerraoui, and L. Rodrigues. "Introduction to Reliable and Secure Distributed Programming". (Second Edition). Springer, 2011.

[13] V. Hadzilacos and S. Toueg. "Fault-tolerant broadcasts and related problems.". In S. J. Mullender, editor, Distributed Systems (2nd Ed.). New York, 1993.

[14] Lamport, L., Shostak, R., and Pease, M. "The Byzantine generals problem". ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3), 382-401, 1982.

[15] Lamport, L. "The part-time parliament". ACM Transactions on Computer Systems, 16(2):133169, May 1998.

[16] Lamport, L. "Paxos made simple". SIGACT News, 32(4):5158, 2001.

[17] B. M. Oki and B. Liskov. "Viewstamped replication: A new primary copy method to support highly- available distributed systems". In Proc. 7th ACM Symposium on Principles of Distributed Computing (PODC), pages 817, 1988.

[18] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. "ZooKeeper: Wait-free coordination for internet- scale systems". In Proc. USENIX Annual Technical Conference, 2010.

[19] D. Ongaro and J. K. Ousterhout. "In search of an understandable consensus algorithm". In Proc. USENIX Annual Technical Conference, pages 305319, 2014.

[20] M. Castro and B. Liskov. "Practical Byzantine fault tolerance and proactive recovery". ACM Transactions on Computer Systems, 20(4):398461, Nov. 2002.

[21] Bessani, A., Sousa, J., and Alchieri, E. E. "State machine replication for the masses with BFT-SMaRt". In 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 355-362, June 2014.

[22] C. Dwork, N. Lynch, and L. Stockmeyer "Consensus in the presence of partial synchrony". Journal of the ACM (JACM), 35(2):288323, 1988.

[23] M. J. Fischer, N. A. Lynch, and M. S. Paterson. "Impossibility of distributed consensus with one faulty process". Journal of the ACM (JACM), 32(2):374382, 1985.

[24] Brewer, E. A. "Towards robust distributed systems.". In PODC (Vol. 7), July 2000.

[25] M. J. M. Chowdhury and M. S. Ferdous and K. Biswas and N. Chowdhury and A. S. M. Kayes and M. Alazab and P. Watters "A Comparative Analysis of Distributed Ledger Technology Platforms". IEEE Access, 7:167930-167943, 2019.

[26] Iot, Joi "The Fintech Bubble". 2018 [Online] Available: https://joi.ito.com/weblog/2016/06/14/-the-fintech-bu.html. June 14, 2016. Accessed on May 12, 2019

[27] Xiao, David "The Four Layers of the Blockchain". 2018 [Online] Available: https://medium.com/@coriacetic/the-four-layers-of-the-blockchain-dc1376efa10f. June 22, 2016. Accessed on April 12, 2019

[28] Namecoin. 2018 [Online] Available: https://namecoin.org/. Accessed on May 20, 2019.

[29] Proof of Existence. 2018 [Online] Available: https://proofofexistence.com. Accessed on May 20, 2019.

[30] Fromknecht, C., Velicanu, D., and Yakoubov, S. CertCoin: A NameCoin Based Decentralized Authentication System. May 14, 2014. 6.857 Unpublished class project. 2018 [Online] Available: http://courses.csail.mit.edu/6.857/2014/files/19-fromknecht-velicann-yakoubov-certcoin.pdf. Accessed on May 20, 2019.

[31] Ferdous, Md Sadek and Biswas, Kamanashis and Chowdhury, Mohammad Jabed Morshed and Chowdhury, Niaz and Muthukku-

marasamy, Vallipuram "Integrated platforms for blockchain enablement". Advances in Computers, Elsevier, 2019.
[32] Seigen, Max Jameson, Tuomo Nieminen, Neocortex and Antonio M. Juarez "CryptoNight Hash Function". March, 2013. [Online] Available: https://cryptonote.org/cns/cns008.txt.
[33] Dwork, Cynthia and Naor, Moni "Pricing via processing or combatting junk mail". Annual International Cryptology Conference, 139–147, 1992.
[34] Douceur, J. R. "The sybil attack". In International workshop on peer-to-peer systems (pp. 251-260), 2002.
[35] Bentov, Iddo and Lee, Charles and Mizrahi, Alex and Rosenfeld, Meni "Proof of Activity: Extending Bitcoin's Proof of Work via Proof of Stake [Extended Abstract]". SIGMETRICS Perform. Eval. Rev., Volume 43, issue 3.
[36] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. "The Honey Badger of BFT Protocols." In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). ACM, New York, NY, USA, 31-42. DOI: https://doi.org/10.1145/2976749.2978399
[37] Athanasios N. Nikolakopoulos and John D. Garofalakis. "NCDawareRank: a novel ranking method that exploits the decomposable structure of the web." In Proceedings of the sixth ACM international conference on Web search and data mining (WSDM '13). ACM, New York, NY, USA, 143-152.
[38] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov and Krzysztof Pietrzak "Proofs of Space". Cryptology ePrint Archive, Report 2013/796. [Online] Available: https://eprint.iacr.org/2013/796
[39] Ren, Ling and Devadas, Srinivas "Proof of Space from Stacked Expanders" Proceedings, Part I, of the 14th International Conference on Theory of Cryptography - Volume 9985, 2016
[40] Sunoo Park, Krzysztof Pietrzak, Albert Kwon, Joë l Alwen and Georg Fuchsbauer and Peter Gaži "SpaceMint: A Cryptocurrency Based on Proofs of Space" Cryptology ePrint Archive, Report 2015/528 [Online] Available: https://eprint.iacr.org/2015/528
[41] Hamza Abusalah and Joël Alwen and Bram Cohen and Danylo Khilko and Krzysztof Pietrzak and Leonid Reyzin "Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space" Cryptology ePrint Archive, Report 2017/893 [Online] Available: https://eprint.iacr.org/2017/893
[42] Weichbrodt, Nico and Kurmus, Anil and Pietzuch, Peter and Kapitza, Rüdiger "Computer Security ESORICS 2016" Pages 440–457 Springer International Publishing
[43] Brasser F, Müller U, Dmitrienko A, Kostiainen K, Capkun S, Sadeghi A-R "Software Grand Exposure: SGX Cache Attacks Are Practical" In proceedings of the 11th USENIX Workshop on Offensive Technologies (WOOT 17) USENIX Association
[44] Keir Finlow-Bates "A Lightweight Blockchain Consensus Protocol". August, 2017. [Online] Available: http://www.chainfrog.com/wp-content/uploads/2017/08/consensus.pdf.
[45] Back, Adam Hashcash-a denial of service counter-measure. 2002.
[46] Bertoni, Guido and Daemen, Joan and Peeters, Michaël and Van Assche, Gilles "Keccak" In Annual International Conference on the Theory and Applications of Cryptographic Techniques pp. 313–314, 2013.
[47] Percival, C. and Josefsson, S. "The scrypt Password-Based Key Derivation Function". [Online] Available: https://tools.ietf.org/html/rfc7914 August, 2016.
[48] "Wikipedia entry on Custon Hardware Attack". [Online] Available: https://en.wikipedia.org/wiki/Custom_hardware_attack Accessed on 21 May, 2019.
[49] Bernstein, Daniel J "The Salsa20 family of stream ciphers" New stream cipher designs pp. 84–97, Springer, 2008.
[50] "SThe Scrypt Mining Algorithm: Everything You Need to Know". [Online] Available: https://www.easypc.io/crypto-mining/scrypt-hardware/ Accessed on 21 May, 2019.
[51] Buntinx, JP "Scrypt vs X11 vs SHA-256". [Online] Available: https://themerkle.com/scrypt-vs-x11-vs-sha-256/ March 23, 2017.
[52] Bernstein, Daniel J ChaCha, a variant of Salsa20 "New stream cipher designs" Vol. 8, pp. 3–5, 2008.
[53] "Announcement of Aiden - Scrypt-OG crypto-currency". [Online] Available: https://bitcointalk.org/index.php?topic=558414.0 Accessed on July 24, 2019.
[54] "Cryptocurrency Mining Hash Algorithms". http://www.bitcoinlion.com/cryptocurrency-mining-hash-algorithms/ [Online] Available: July 24, 2019. Accessed on July 24, 2019.
[55] Biryukov, Alex and Khovratovich, Dmitry "Equihash: Asymmetric proof-of-work based on the generalized birthday problem". Ledger(2). 2017.
[56] Wagner, David "A generalized birthday problem". Cryptor(2442). pages 288–303. 2002.
[57] "Crypto-currencies utilising Equihash". [Online] Available: https://www.coingecko.com/en?hashing_algorithm=Equihash Accessed on July 24, 2019.
[58] "Ethash". [Online] Available: https://github.com/ethereum/wiki/wiki/Ethash Accessed on July 24, 2019.
[59] "Dagger-Hashimoto". [Online] Available: https://github.com/ethereum/wiki/blob/master/Dagger-Hashimoto.md Accessed on July 24, 2019.
[60] Buterin, Vitalik "Dagger: A Memory-Hard to Compute, Memory-Easy to Verify Scrypt Alternative". http://www.hashcash.org/papers/dagger.html Accessed on July 24, 2019.
[61] Lerner, Sergio "Ethereum Dagger PoW function is flawed". [Online] Available: https://bitslog.wordpress.com/2014/01/17/ethereum-dagger-pow-is-flawed/ 17 January, 2017. Accessed on July 24, 2019.
[62] "Crypto-currencies utilising Ethash". [Online] Available: https://www.coingecko.com/en?hashing_algorithm=Ethash Accessed on July 24, 2019.
[63] "Crypto-currencies utilising Dagger". [Online] Available: https://www.coingecko.com/en?hashing_algorithm=Dagger Accessed on July 24, 2019.
[64] Gligoroski, Danilo and Klima, Vlastimil and Knapskog, Svein Johan and El-Hadedy, Mohamed and Amundsen, Jørn "Cryptographic hash function blue midnight wish" In Proceedings of the 1st International Workshop on Security and Communication Networks pp. 1–8, 2009
[65] Gauravaram, Praveen and Knudsen, Lars R and Matusiewicz, Krystian and Mendel, Florian and Rechberger, Christian and Schläffer, Martin and Thomsen, Søren S "Grøstl-a SHA-3 candidate" Dagstuhl Seminar Proceedings 2009
[66] Wu, Hongjun "The hash function JH" Submission to NIST (round 3) Vol. 6, 2011
[67] "Quark Coin". [Online] Available: http://www.quarkcoins.com/ Accessed on July 24, 2019.
[68] "First Impressions from the Baikal Mini Miner ASIC". [Online] Available: https://cryptomining-blog.com/tag/quark-asic-miner/ July 24, 2019 Accessed on July 24, 2019.
[69] "Bitcoin". [Online] Available: http://www.bitcoin.org/ Accessed on April 24, 2019.
[70] "Bitcoin Cash". [Online] Available: https://www.bitcoincash.org/ Accessed on July 24, 2019.
[71] "Syscoin". [Online] Available: http://syscoin.org/ Accessed on July 24, 2019.
[72] "Peer Coin". [Online] Available: https://peercoin.net Accessed on July 24, 2019.
[73] "Counterparty". [Online] Available: https://counterparty.io Accessed on July 24, 2019.
[74] Aumasson, Jean-Philippe and Neves, Samuel and Wilcox-OHearn, Zooko and Winnerlein, Christian "BLAKE2: simpler, smaller, fast as MD5" International Conference on Applied Cryptography and Network Security. pp. 119–135, 2013.
[75] "Emercoin. [Online] Available: https://emercoin.com/ Accessed on July 24, 2019.
[76] "Namecoin". [Online] Available: https://namecoin.org/ Accessed on July 24, 2019.
[77] "Steem Dollars". [Online] Available: https://steem.io/ Accessed on July 24, 2019.
[78] "Crown". [Online] Available: https://crown.tech/ Accessed on July 24, 2019.
[79] "Ommi (Mastercoin)". [Online] Available: http://www.omnilayer.org/ Accessed on July 24, 2019.
[80] "Litecoin". [Online] Available: http://www.omnilayer.org/ Accessed on July 24, 2019.
[81] "Bitconnect". [Online] Available: https://bitconnect.co/ Accessed on May 24, 2019.
[82] "Verge". [Online] Available: https://vergecurrency.com Accessed on July 20, 2019.
[83] "Bitmark". [Online] Available: https://bitmark.com/ Accessed on July 20, 2019.
[84] "Dogecoin". [Online] Available: http://dogecoin.com/ Accessed on July 20, 2019.

[85] "Gamecredit)". [Online] Available: https://gamecredits.com/ Accessed on July 20, 2019.
[86] "Einsteinnium". [Online] Available: https://www.emc2.foundation/ Accessed on July 20, 2019.
[87] "Voxels". [Online] Available: http://thevoxel.com/ Accessed on July 20, 2019.
[88] "Viacoin". [Online] Available: https://viacoin.org/ Accessed on July 10, 2019.
[89] "Hempcoin". [Online] Available: http://hempcoin.org/ Accessed on July 10, 2019.
[90] "Zcash". [Online] Available: https://z.cash/ Accessed on July 10, 2019.
[91] "Bitcoin Gold". [Online] Available: https://bitcoingold.org/ Accessed on July 10, 2019.
[92] "Komodo". [Online] Available: https://komodoplatform.com/en Accessed on July 10, 2019.
[93] "Zclassic". [Online] Available: http://zclassic.org/ Accessed on July 10, 2019.
[94] "ZenCash". [Online] Available: https://zensystem.io/ Accessed on July 10, 2019.
[95] "Hush". [Online] Available: https://myhush.org/ Accessed on July 10, 2019.
[96] "bitcoinz". [Online] Available: https://btcz.rocks/en/ Accessed on July 10, 2019.
[97] "Vote Coin". [Online] Available: https://votecoin.site/ Accessed on July 10, 2019.
[98] "Ethereum". [Online] Available: https://www.ethereum.org/ Accessed on July 10, 2019.
[99] "Ethereum Classic". [Online] Available: https://ethereumclassic.github.io/ Accessed on July 10, 2019.
[100] "Ubiq". [Online] Available: https://ubiqsmart.com/ Accessed on July 10, 2019.
[101] "Shif". [Online] Available: http://www.shiftnrg.org/ Accessed on July 10, 2019.
[102] "Expanse". https://www.expanse.tech/ Accessed on July 10, 2019.
[103] "DubaiCoin". [Online] Available: http://www.arabianchain.org/ Accessed on July 10, 2019.
[104] "SOILcoin". [Online] Available: https://github.com/Soilcoin Accessed on July 10, 2019.
[105] "Krypton". [Online] Available: http://krypton.rocks/ Accessed on July 10, 2019.
[106] "Red Pulse". [Online] Available: https://www.red-pulse.com/landing Accessed on July 10, 2019.
[107] "Feathercoin". [Online] Available: https://www.feathercoin.com/ Accessed on July 10, 2019.
[108] "GoByte". [Online] Available: https://gobyte.network/ Accessed on July 10, 2019.
[109] "UFO Coin". [Online] Available: https://ufocoin.net/ Accessed on July 10, 2019.
[110] "Innova". [Online] Available: https://innovacoin.info/ Accessed on July 10, 2019.
[111] "rowdCoin". [Online] Available: http://crowdcoin.site/ Accessed on July 10, 2019.
[112] "Vivo". [Online] Available: https://www.vivocrypto.com/ Accessed on July 10, 2019.
[113] "Desire". [Online] Available: https://www.desire-crypto.com/ Accessed on July 10, 2019.
[114] "Orbitcoin". [Online] Available: https://www.orbitcoin.org/ Accessed on July 10, 2019.
[115] "Phoenixcoin". [Online] Available: http://phoenixcoin.org/ Accessed on July 10, 2019.
[116] "Bitcore". [Online] Available: https://bitcore.cc/ Accessed on July 10, 2019.
[117] "Dash". [Online] Available: https://www.dash.org/ Accessed on July 10, 2019.
[118] "Stratis. [Online] Available: https://stratisplatform.com/ Accessed on July 10, 2019.
[119] "Cloakcoin". [Online] Available: https://www.cloakcoin.com/ Accessed on July 10, 2019.
[120] "Stealthcoin". [Online] Available: https://www.stealthcoin.com/ Accessed on July 10, 2019.
[121] "DeepOnion". [Online] Available: https://deeponion.org/ Accessed on July 10, 2019.
[122] "HTMLcoin". [Online] Available: https://htmlcoin.com/ Accessed on July 10, 2019.
[123] "Regal Coin". [Online] Available: https://regalcoin.co/ Accessed on July 10, 2019.
[124] "Memetic". [Online] Available: https://memetic.ai/ Accessed on July 10, 2019.
[125] "Exclusive Coin". [Online] Available: https://exclusivecoin.pw/ Accessed on July 10, 2019.
[126] "Creditbit". [Online] Available: https://www.creditbit.org/ Accessed on July 10, 2019.
[127] "Quark". [Online] Available: http://www.qrknet.info/ Accessed on July 10, 2019.
[128] "PIVX". [Online] Available: https://pivx.org/ Accessed on July 10, 2019.
[129] "MonetaryUnit". [Online] Available: https://www.monetaryunit.org/ Accessed on July 10, 2019.
[130] "ALQO". [Online] Available: https://alqo.org/ Accessed on July 10, 2019.
[131] "Bitcloud". [Online] Available: https://bit-cloud.info/ Accessed on July 10, 2019.
[132] "Zurcoin". [Online] Available: http://zurcoin.org/ Accessed on July 10, 2019.
[133] "AmsterdamCoin". [Online] Available: https://amsterdamcoin.com/ Accessed on July 10, 2019.
[134] "Animecoin". [Online] Available: https://github.com/testzcrypto/Animecoin/releases/tag/0.8.3.2 Accessed on July 10, 2019.
[135] "Vertcoin". [Online] Available: https://vertcoin.org/ Accessed on July 10, 2019.
[136] "Monacoin". [Online] Available: https://monacoin.org/ Accessed on July 10, 2019.
[137] "Crypto". [Online] Available: http://tailflick.wixsite.com/official-crypto Accessed on July 10, 2019.
[138] "Quark Coin Wiki". [Online] Available: http://coinwiki.info/en/Quark Accessed on July 10, 2019.
[139] "Crypto-currencies utilising Quark". [Online] Available: https://www.coingecko.com/en?hashing_algorithm=Quark Accessed on July 10, 2019.
[140] "Lyra2RE-A new PoW algorithm for an ASIC-free future". [Online] Available: https://vertcoin.org/wp-content/uploads/2017/10/Vertcoin_Lyra2RE_Paper_11292014.pdf Accessed on July 10, 2019.
[141] "Wikipedia entry on Vertcoin". [Online] Available: https://en.wikipedia.org/wiki/Vertcoin Accessed on July 10, 2019.
[142] "Crypto-currencies utilising Lyra2RE". [Online] Available: https://www.coingecko.com/en?hashing_algorithm=Lyra2RE Accessed on July 10, 2019.
[143] "Wiki entry on M7". [Online] Available: http://cryptonite.info/wiki/index.php?title=M7_PoW Accessed on July 10, 2019.
[144] "CudaMiner - a multi-threaded GPU miner for Cryptonite". [Online] Available: https://github.com/MiniblockchainProject/CudaMiner Accessed on July 10, 2019.
[145] "Bitcore announcement on Bitcointalk". [Online] Available: https://bitcointalk.org/index.php?topic=1883902.0 Accessed on July 10, 2019.
[146] Doering, John "NeoScrypt, a Strong Memory Intensive Key Derivation Function". http://phoenixcoin.org/archive/neoscrypt_v1.pdf [Online] Available: July 26, 2014. Accessed on July 10, 2019.
[147] "Crypto-currencies utilising NeoScrypt". [Online] Available: https://www.coingecko.com/en?hashing_algorithm=NeoScrypt Accessed on July 10, 2019.
[148] "Bitcoin energy consumption". [Online] Available: https://digiconomist.net/bitcoin-energy-consumption Accessed on July 10, 2019.
[149] "Ethereum energy consumption". [Online] Available: https://digiconomist.net/ethereum-energy-consumption Accessed on July 10, 2019.
[150] "Wikipedia entry on Economies of scale". [Online] Available: https://en.wikipedia.org/wiki/Economies_of_scale Accessed on July 10, 2019.
[151] "Wikipedia entry on Tragedy of the commons". [Online] Available: https://en.wikipedia.org/wiki/Tragedy_of_the_commons Accessed on July 10, 2019.
[152] "Bitcoin hashrate distribution". [Online] Available: https://blockchain.info/pools Accessed on July 10, 2019.
[153] Eyal, I. and Sirer, E.G. "Majority is not enough: Bitcoin mining is vulnerable". International Conference on Financial Cryptography and Data Security pages 436–454. March, 2014.

[154] QuantumMechanic "Proof of stake instead of proof of work". [Online] Available: https://bitcointalk.org/index.php?topic=27787.0 July 11, 2011. Accessed on May 11, 2019.

[155] "Proof of Stake FAQ". [Online] Available: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ Accessed on August 5, 2019.

[156] Choi, Jon "Ethereum Casper 101". [Online] Available: https://medium.com/@jonchoi/ethereum-casper-101-7a851a4f1eb0 October 22, 2017. Accessed on July 10, 2019.

[157] "Proof of Stake versus Proof of Work". [Online] Available: http://bitfury.com/content/5-white-papers-research/pos-vs-pow-1.0.2.pdf September 13, 2015. Accessed on August 5, 2019.

[158] "Consensus Compare: Casper vs. Tendermint". [Online] Available: https://blog.cosmos.network/consensus-compare-casper-vs-tendermint-6df154ad56ae November 16, 2017. Accessed on July 10, 2019.

[159] Zamfir, Vlad "The History of Casper - Chapter 4". [Online] Available: https://medium.com/@Vlad_Zamfir/the-history-of-casper-chapter-4-3855638b5f0e December 12, 2016. Accessed on July 10, 2019.

[160] "Gridcoin". [Online] Available: https://github.com/gridcoin-community/Gridcoin-Wiki/wiki Accessed on June 22, 2019.

[161] "Gridcoin Whitepaper. [Online] Available: https://gridcoin.us/assets/img/whitepaper.pdf Accessed on 24 June, 2019.

[162] "Wikipedia entry on Berkeley Open Infrastructure for Network Computing (BOINC)". [Online] Available: https://en.wikipedia.org/wiki/Berkeley_Open_Infrastructure_for_Network_Computing Accessed on 21 June, 2019.

[163] "Proof of Burn". [Online] Available: https://en.bitcoin.it/wiki/Proof_of_burn Accessed on 21 June, 2019.

[164] "Slimcoin". [Online] Available: http://slimco.in/ Accessed on 24 June, 2019.

[165] "Slimcoin Whitepaper". [Online] Available: https://www.doc.ic.ac.uk/~ids/realdotdot/crypto_papers_etc_worth_reading/proof_of_burn/slimcoin_whitepaper.pdf Accessed on 24 June, 2019.

[166] "Reddcoin". [Online] Available: https://www.reddcoin.com/ Accessed on 24 June, 2019.

[167] "Reddcoin Whitepaper. [Online] Available: https://www.reddcoin.com/papers/PoSV.pdf Accessed on 24 June, 2019.

[168] "Reddcoin Wiki". [Online] Available: https://wiki.reddcoin.com/Proof_of_Stake_Velocity_(PoSV) Accessed on 24 June, 2019.

[169] "The Velocity of Money for Beginners". [Online] Available: https://www.joshuakennon.com/the-velocity-of-money-for-beginners/ Accessed on 25 June, 2019.

[170] "Faircoin Crypto-currency". [Online] Available: https://fair-coin.org Accessed on 24 June, 2019.

[171] "Faircoin Whitepaper, Version 1.2". [Online] Available: https://fair-coin.org/sites/default/files/FairCoin2_whitepaper_V1.2.pdf July, 2018. Accessed on 24 June, 2019.

[172] "NEM Technical Reference, Version 1.2.1". [Online] Available: https://nem.io/wp-content/themes/nem/files/NEM_techRef.pdf February 23, 2018. Accessed on 24 June, 2019.

[173] "Faircoin FAQ". [Online] Available: https://fair-coin.org/en/faircoin-faqs Accessed on 24 June, 2019.

[174] King, Sunny and Nadal, Scott "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake". [Online] Available: https://peercoin.net/assets/paper/peercoin-paper.pdf August 19, 2012. Accessed on May 5, 2019.

[175] "Peercoin discussion forum, discussion#3043". [Online] Available: https://talk.peercoin.net/t/ppc-switching-between-pos-and-pow/3043 November 4, 2014. Accessed on May 5, 2019.

[176] Bentov, Iddo and Gabizon, Ariel and Mizrahi, Alex "Cryptocurrencies without proof of work". International Conference on Financial Cryptography and Data Security pages 142–157. 2016

[177] "Decred Platform". [Online] Available: https://decred.org/ Accessed on June 23, 2019.

[178] "Tendermint introduction". [Online] Available: http://tendermint.readthedocs.io/projects/tools/en/master/introduction.html Accessed on July 30, 2019.

[179] Kwon, Jae "Tendermint: Consensus without Mining". [Online] Available: https://tendermint.com/static/docs/tendermint.pdf 2014. Accessed on July 30, 2019.

[180] "Tendermint wiki". [Online] Available: https://github.com/tendermint/tendermint/wiki/Byzantine-Consensus-Algorithm Accessed on July 30, 2019.

[181] Buterin, Vitalik and Griffith, Virgil "Casper the Friendly Finality Gadget". [Online] Available: https://github.com/ethereum/research/blob/master/papers/casper-basics/casper_basics.pdf Accessed on August 1, 2019.

[182] Zamfir, Vlad "Casper the Friendly Ghost-A Correct-by-Construction Blockchain Consensus Protocol". [Online] Available: https://github.com/ethereum/research/blob/master/papers/CasperTFG/CasperTFG.pdf Accessed on August 1, 2019.

[183] Sompolinsky, Y. and Zohar, A. "Secure high-rate transaction processing in bitcoin". International Conference on Financial Cryptography and Data Security pp. 507-527. January, 2015.

[184] JP Buntinx "What is Delegated Proof-of-Stake?". [Online] Available: https://themerkle.com/what-is-delegated-proof-of-stake/ April 20, 2017. Accessed on August 1, 2019.

[185] Kiayias, A., Russell, A., David, B., and Oliynykov, R. "Ouroboros: A provably secure proof-of-stake blockchain protocol". Annual International Cryptology Conference pp. 357-388. August, 2017.

[186] "OUROBOROS PROOF OF STAKE ALGORITHM". [Online] Available: https://cardanodocs.com/cardano/proof-of-stake/ Accessed on August 2, 2019.

[187] "Cardano Platform". [Online] Available: https://www.cardanohub.org/en/home/ Accessed on August 2, 2019.

[188] "EOS Platform". [Online] Available: https://eos.io/ Accessed on 24 May, 2019.

[189] "EOS Whitepaper". [Online] Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md Accessed on 24 May, 2019.

[190] Kim, Jonathan "EOS Raises Record-Breaking $4 Billion from Crowdsale". [Online] Available: https://cryptoslate.com/eos-raises-record-breaking-4-billion-from-crowdsale/ 29 May, 2018. Accessed on 24 May, 2019.

[191] "What is EOS? Most Comprehensive Guide Part 2". [Online] Available: https://blockgeeks.com/guides/what-is-eos-part-2/ Accessed on 24 May, 2019.

[192] Rosic, Ameer "What is EOS Blockchain: Beginners Guide". [Online] Available: https://blockgeeks.com/guides/eos-blockchain/ Accessed on 24 May, 2019.

[193] "Tron Platform". [Online] Available: https://tron.network/ Accessed on 24 May, 2019.

[194] "Tron Whitepaper". [Online] Available: https://tron.network/static/doc/white_paper_v_2_0.pdf Accessed on 24 May, 2019.

[195] "TRONs Consensus, "How it works"". [Online] Available: https://medium.com/@TRONNews/trons-consensus-how-it-works-6fd231b63715 11 November, 2018. Accessed on 24 May, 2019.

[196] "Tezos Platform". [Online] Available: https://tezos.com/ Accessed on 24 May, 2019.

[197] Goodman, L.M. "Tezos Whitepaper". [Online] Available: https://tezos.com/static/white_paper-2dc8c02267a8fb86bd67a108199441bf.pdf 2 September, 2014. Accessed on 24 May, 2019.

[198] Arluck, Jacob "Liquid Proof-of-Stake". [Online] Available: https://medium.com/@TRONNews/trons-consensus-how-it-works-6fd231b63715 31 July, 2018. Accessed on 24 May, 2019.

[199] Arluck, Jacob "Overview of Tezos Economics Model". [Online] Available: https://www.infinitystones.io/tezos/ Accessed on 24 May, 2019.

[200] "Lisk Platform". [Online] Available: https://lisk.io/ Accessed on 1 June, 2019.

[201] "Lisk Whitepaper". [Online] Available: https://github.com/slasheks/lisk-whitepaper/blob/development/LiskWhitepaper.md Accessed on 1 June, 2019.

[202] "Ark Platform". [Online] Available: https://ark.io/ Accessed on 1 June, 2019.

[203] "Ark Whitepaper". [Online] Available: https://ark.io/Whitepaper.pdf Accessed on 1 June, 2019.

[204] Ateniese, Giuseppe and Bonacina, Ilario and Faonio, Antonio and Galesi, Nicola "Proofs of Space: When Space Is of the Essence". In proceedings of 9th International Conference on Security and Cryptography for Networks: , SCN 2014, Amalfi, Italy, September 3-5, 2014.

[205] "Peercoin discussion forum, discussion#2524". [Online] Available: https://talk.peercoin.net/t/the-complete-guide-to-minting/2524 June 15, 2014. Accessed on May 5, 2019.
[206] "Hyperledger". [Online] Available: https://www.hyperledger.org/ Accessed on July 10, 2019.
[207] "Hyperledger Fabric". [Online] Available: https://www.hyperledger.org/projects/fabric Accessed on July 10, 2019.
[208] "Hyperledger Sawtooth". [Online] Available: https://www.hyperledger.org/projects/sawtooth Accessed on July 10, 2019.
[209] "Hyperledger Burrow". [Online] Available: https://www.hyperledger.org/projects/hyperledger-burrow Accessed on July 10, 2019.
[210] "Hyperledger Iroha". [Online] Available: https://www.hyperledger.org/projects/iroha Accessed on July 10, 2019.
[211] "Hyperledger Indy". [Online] Available: https://www.hyperledger.org/projects/hyperledger-indy Accessed on July 10, 2019.
[212] "Apache Kafka". [Online] Available: https://kafka.apache.org Accessed on July 10, 2019.
[213] "Apache Zookeeper". [Online] Available: https://zookeeper.apache.org/ Accessed on July 10, 2019.
[214] "Monax Industries Limited". [Online] Available: https://monax.io/ Accessed on July 10, 2019.
[215] "Iroha Codebase". [Online] Available: https://github.com/hyperledger/iroha Accessed on July 10, 2019.
[216] Palanivel, C. "Hyperledger Iroha - Architecture, Functional/Logical Flow & Consensus(YAC) Mechanism". [Online] Available: https://www.linkedin.com/pulse/hyperledger-iroha-architecture-functionallogical-chandrasekaran/ April 30, 2018. Accessed on July 10, 2019.
[217] "Hyperledger Architecture, Volume 1". [Online] Available: https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf August, 2017. Accessed on July 10, 2019.
[218] "gRPC". [Online] Available: https://grpc.io/ Accessed on July 10, 2019.
[219] "Iroha Documentation". [Online] Available: https://iroha.readthedocs.io/en/latest/index.html Accessed on July 10, 2019.
[220] Chen, L., Xu, L., Shah, N., Gao, Z., Lu, Y., & Shi, W. "On Security Analysis of Proof-of-Elapsed-Time (PoET)". In International Symposium on Stabilization, Safety, and Security of Distributed Systems, pp. 282-297, November, 2017.
[221] "Burrow Codebase on Github". [Online] Available: https://github.com/hyperledger/burrow Accessed on July 10, 2019.
[222] Ferdous, M.S., Chowdhury, F. and Alassafi, M. "In Search of Self-Sovereign Identity Leveraging Blockchain Technology". IEEE Access, 7, pp.103059-103079, 2019.
[223] "Sovrin Foundation". [Online] Available: https://sovrin.org/ Accessed on March 27, 2018.
[224] Aublin, P. L., Mokhtar, S. B., and Quma, V. "Rbft: Redundant byzantine fault tolerance". In IEEE 33rd International Conference on Distributed Computing Systems (ICDCS), 2013, pp. 297-306, July 2013.
[225] "Iroha Consensus Discussion". [Online] Available: https://github.com/hyperledger/indy-plenum/blob/master/docs/main.md Accessed on July 10, 2019.
[226] Dogan, T. "Who Scales It Best? Blockchains' TPS Analysis". [Online] Available: https://hackernoon.com/who-scales-it-best-blockchains-tps-analysis-pv39g25mg Accessed on July 27, 2019.
[227] O'Neal, S. "Who Scales It Best? Inside Blockchains Ongoing Transactions-Per-Second Race". [Online] Available: https://cointelegraph.com/news/who-scales-it-best-inside-blockchains-ongoing-transactions-per-second-race. January 22, 2019. Accessed on July 27, 2019.
[228] LeMahieu, C. "Nano: A Feeless Distributed Cryptocurrency Network". [Online] Available: https://nano.org/en/whitepaper. Accessed on July 27, 2019.
[229] Popov, S. "The Tangle". [Online] Available: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1_4_3.pdf. April 30, 2018. Accessed on July 27, 2019.

Layer 2 Blockchain Scaling: a Survey

Cosimo Sguanci
cosimo.sguanci@mail.polimi.it

Roberto Spatafora
roberto.spatafora@mail.polimi.it

Andrea Mario Vergani
andreamario.vergani@mail.polimi.it

June 15, 2021
Version 1.0

Abstract

Blockchain technology is affected by massive limitations in scalability with consequent repercussions on performance. This discussion aims at analyzing the state of the art of current available Layer II solutions to overcome these limitations, both focusing on theoretical and practical aspects and highlighting the main differences among the examined frameworks. The structure of the work is based on three major sections. In particular, the first one is an introductory part about the technology, the scalability issue and Layer II as a solution. The second section represents the core of the discussion and consists of three different subsections, each with a detailed examination of the respective solution (Lightning Network, Plasma, Rollups); the analysis of each solution is based on how it affects five key aspects of blockchain technology and Layer II: scalability, security, decentralization, privacy, fees and micropayments (the last two are analyzed together given their high correlation). Finally, the third section includes a tabular summary, followed by a detailed description of a use-case specifically thought for a practical evaluation of the presented frameworks. The results of the work met expectations: all solutions effectively contribute to increasing scalability. A crucial clarification is that none of the three dominates the others in all possible fields of application, and the consequences in adopting each, are different. Therefore, the choice depends on the application context, and a trade-off must be found between the aspects previously mentioned.

1 Blockchain Introduction

In its most basic meaning, Blockchain technology consists of a decentralized public ledger capable of keeping a record of immutable (and linked) information. However, the original goal of this technology, as stated in the original Bitcoin white paper written by Satoshi Nakamoto, is to provide a trustless electronic payment system that does not rely on any third-party authority to manage disputes between actors. The trust, which is needed in traditional cash systems, can be replaced by cryptographic proofs.

Such a system can be implemented as a *peer-to-peer* network of nodes, in which a certain number of transactions are periodically bundled into blocks. Blocks are hashed to generate timestamps and every block includes the hash of the previous block. This mechanism forms a chain of timestamped information that, in this context, is represented by transactions. All the generated blocks are propagated across the network and locally stored by each node; in this way every node can act as a *validator* of transactions, and the so-called *double spending* problem can be solved.

To allow *peer-to-peer* direct payments, transactions must be computationally infeasible to reverse: this is allowed by the *proof-of-work* mechanism, in which every node interested in generating new blocks (also called *miners*) search for a value that, when hashed using a particular hash algorithm, has certain properties. For instance, as in the Bitcoin protocol, this could involve the fact that the resulting hash must have a certain number of leading zeros. *Miners* are then incentivized to sustain the network by collecting the fees that users spend to execute transactions, thus improving the global *hashrate* of the Blockchain.

Thanks to this paradigm, combined with the fact that nodes consider the longest chain to be the valid state, once a block has been generated and the computation has been performed, changing the block would require redoing the *proof-of-work* for both the specific block and all subsequent ones.

This system has proven to be secure as soon as honest nodes hold the majority of the network computational power: in this case, honest participants will generate the longest chain, which will be accepted as the true state of the Blockchain, not allowing attackers to create malicious blocks.

It is common to identify three different generations of Blockchains:

- First generation: Blockchains that allow decentralized monetary system and ledger of transactions. An example is the *Bitcoin* Blockchain.
- Second generation: Blockchains with support for running *smart contracts* (e.g. *Ethereum*). These Blockchains are decentralized platforms which can be used to run programs in a decentralized fashion, to achieve complex functionalities without the need for a central authority (such as an Application Server).

- Third generation: Blockchains that are exploring different *consensus algorithms* other than *proof-of-work* (one example is the *proof-of-stake* mechanism). They generally also aim at substantially improving the scalability of pre-existing Blockchains. Examples of this type of Blockchain are *Cardano*, *Polkadot*.

2 Blockchain Scalability issue

A critical aspect of the blockchain concerns its scalability. At the moment of writing, scalability is considered the bottleneck of the blockchain infrastructure. The aforementioned could potentially compete with the largest electronic payment circuits. However, it is limited by being able to handle few transactions per second (TPS). To clarify the problem, it is sufficient to report the two best-known blockchains as an example: Bitcoin processes 4.6 TPS and Ethereum processes around 14.3 TPS (slightly variable values), while one of the largest electronic payment circuits, Visa, processes around 1,736 TPS (and has been able to reach peaks of 47,000 TPS).
Currently, the Bitcoin blockchain generates a new block every about 10 minutes (Time Block generation, TB) and, the block size (B) in the chain is 1MB (1,048,576 Bytes). The average transaction size is 380 Bytes. Therefore, the number of transactions that fit into a Bitcoin block (TPB) is:

$$TPB = \frac{Block\ Size}{Average\ Transaction\ Size} = \frac{1,048,576\ Bytes}{380\ Bytes} \approx 2,759\ transactions$$

As a consequence of the TB and TPB, the number of transactions per second is:

$$TPS = \frac{TPB}{TB} \approx \frac{2,759\ transactions}{600\ seconds} \approx 4.6\ transactions\ per\ second$$

Contrary to what one might think, the scalability problem cannot be solved by simply modeling its parameters: B and TB. As a matter of fact, considering the parameters modeling, it is necessary to make an important clarification: when generating a new block in the blockchain, a crucial factor to be taken into account is the relay time (TR) needed to broadcast the new block to every node on the network. Therefore, this fact imposes a lower limit on TB, below which it is impossible to go. This TR threshold allows to keep all the nodes in the network constantly updated. Additionally, another problem related to the TR arises when expanding the block size (B). Consequently, an increased quantity of information has to be broadcasted to the network.
Again, a practical example from Bitcoin blockchain is considered in the discussion for simplicity of presentation: in January 2021, around 10,000 nodes were estimated in the Bitcoin network. The average time to propagate a block to 99% of the network

is approximately 14 seconds. Thus, TB cannot fall below the 14 seconds threshold. Otherwise, a new block would be generated before an old block would be received by most of the nodes in the network. The problem related to the size of the block becomes evident when increasing it: 14 seconds as Time Relay would no longer be enough.

In 2017, Segregated Witness's (SegWit) soft fork helped to improve block size (scaled up theoretically to 4MB, practically around 2MB) without changes to the core code. Nevertheless, it still does not improve TPS in a scalable manner.

3 Layer II solutions

As explained in the previous section, blockchain's main problem is scalability; this issue derives from the fact that the technology is driven by a decentralized idea as its core, which makes it difficult to scale since transactions have to be broadcasted to the whole network.

All solutions to the presented issue have to deal with the so-called *scalability trilemma*: improvements on blockchain's scalability have a negative effect either on security or decentralization, or both. Since having a decentralized secure network is one of the pillars of blockchain technology, a right tradeoff needs to be found in order to scale.

The most common and used approach to achieve a scalable blockchain is generally known as "Layer 2": the basic idea is that of building a framework which handles transactions *off-chain* (not on the main chain and, in a certain sense, independently of it), thus reducing the load on the blockchain itself and achieving higher transaction speed. Of course, as hinted above, moving in the direction of scalability, and in particular registering transactions *off-chain*, leads to problems in terms of security and decentralization, which need to be addressed by specific countermeasures.

A Layer 2 solution is a secondary protocol built on-top of an existing blockchain; the idea behind the framework can be of different natures, but the key-concept is that of hosting transactions and reporting only a "summary" of them on the main chain. To better understand the situation, it is necessary to distinguish various kinds of high-level Layer 2 (L2) solutions:

- Channels L2 basically create direct or indirect *off-chain* communication channels between nodes; transactions between "connected" nodes are managed on Layer 2, reporting on the main chain only two of them (the one "opening" the channel and the one which "closes" it). For example, Lightning Network for Bitcoin and Raiden Network for Ethereum are based on state channel L2.

- Sidechains L2 are based on "children" blockchains anchored on the main chain

and running in parallel to it; the idea is, in a sense, quite similar to channels, but the consistent difference is that, in sidechains, *off-chain* transactions run on blockchains (while communication channels are not based on a blockchain). Of course, the advantage of moving transactions to another blockchain, which in principle could suffer from the same scalability issue as the main chain, is the following: sidechains involve less nodes and typically "weight" in a different way the *trilemma* between scalability, security and decentralization (in general, they tend to be less decentralized and faster). In the dedicated section, there will be a distinction between standard sidechains and Plasma, a sidechain architecture that offers more guarantees in terms of decentralization and security. Again, transactions handled by a sidechain are reported *on-chain* with only an "opening" and a "closing" one.

- Rollups L2 lie on the general concept of only *executing* a transaction *off-chain*, but always reporting data about it on the main chain; in practice, while channels and sidechains need to report the "summary" of a set of transactions with just two of them on the blockchain, Rollups broadcast a smaller amount of data (with respect to the usual size of *on-chain* transactions) for every off-chain state update.

It is easy to understand that the main concept of all layer 2 solutions is that of lightening the blockchain, in order to help in scaling up. The consequence of this idea is not only higher transaction speed, but also (in general) lower fees (which is a direct consequence of increased TPS): this means that L2 solutions are appropriate places for micropayments to be performed. Other advantages of L2 are the facts that no modification on the main chain is needed and that *off-chain* management of transactions is, in a sense, independent of "Layer 1"; in reality, dependence is essential in order to register *on-chain* a "summary" of transactions, but, apart from this fact, the blockchain is not aware of what happens on Layer 2.

An honourable mention for what regards blockchain scaling solutions is given to sharding. It is a technique based on partitioning the main chain into subsets of nodes, each responsible for a portion of the whole network: every node processes information belonging only to its shard. This solution certainly goes in the direction of improving scalability: "dividing" the load of the chain among different partitions leads to something similar to separate blockchains, characterized by higher transaction speed (since they are lighter); of course, the main chain is not really divided into smaller chains, because shards are still able to share information; however, sharding clearly goes in the direction of lower security and decentralization, thus enforcing once again the *scalability trilemma*. Sharding can not be properly defined as a Layer 2 solution, since no additional *off-chain* framework is effectively added to the main chain; for

this reason, this technique is generally referred to as "Layer 1 scaling solution", to mean exactly the fact that all transactions are managed on the blockchain itself.

The last remark about L2 frameworks is that different protocols can be combined, on-top of the same main chain, in order to improve scalability as much as possible. This is feasible thanks to the fact that the blockchain is totally unaltered and not affected by the use of Layer 2 solutions.

4 Lightning Network (and references to Raiden Network)[1]

Lightning Network (LN) and Raiden Network (RN) are channel networks that operate on top of a blockchain. They have the main objective of enabling fast *peer-to-peer* transactions. LN and RN functioning is based on state channels: the main idea behind this type of solution is the creation of an *off-chain* communication channel between nodes. The L2 solution is in charge of managing transactions between directly or indirectly connected nodes, therefore reducing the main chain's workload, which only has to track the channel's opening and closing transactions. Thus, it allows transactions among nodes in the network which update the tentative distribution of the channel's funds without broadcasting those to the blockchain. LN was introduced as a Layer 2 solution for the Bitcoin blockchain, but it has been integrated also by other blockchains (*e.g.*, Litecoin)[2]; on the other side, RN is Lightning Network's counterpart allowing to perform *off-chain* payments for Ethereum.

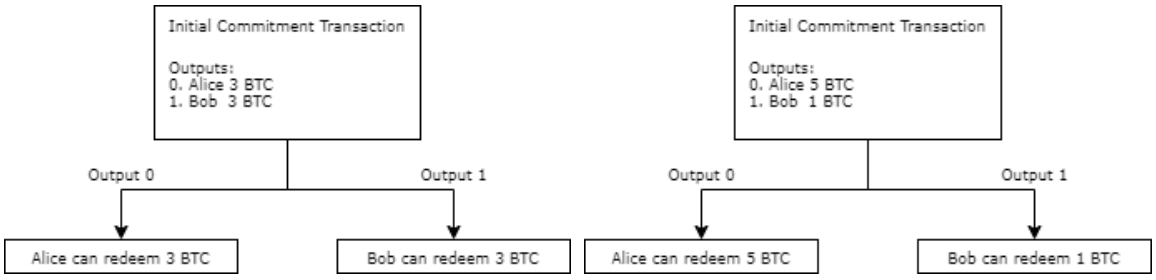
Lightning Network has a separate structure from the main chain, with which it communicates only special opening and closing channel transactions. The opening transaction creates a "channel" between two nodes in the network. Once there is a channel, all the transactions between the two parties are stored in a "private ledger". Only the two counterparties can access the ledger, and neither party can cheat on the other one. The impossibility of cheating is a significant aspect that will be better

[1]Raiden Network is an in-development technology, with less than 65 open channels at the moment of writing. For this reason, this section is mainly focused on the description of Lightning Network. For what regards Raiden Network, descriptions and hints are provided only for the sake of completeness and/or in case of difference with respect to Lightning Network functioning; instead, when nothing else is mentioned, the reader can deduce a very similar behaviour for both Lightning Network and Raiden Network. Since Lightning Network is going to be described also in relation to its actual topology, the reasonable assumption (if not differently specified) is that Raiden Network is likely to evolve in a similar manner in its near future; however, nothing more than assumptions can be carried on at the moment of writing under this point of view, given Raiden Network immaturity.

[2]From this moment on, Lightning Network description is focused on its use as Bitcoin blockchain Layer 2 solution.

analyzed in the discussion.

To clearly understand the use of the channel, let's imagine two users, Alice and Bob, who opened one. Each of them put 3 BTC in a "contract" with the counterparty. After ten transactions, a possible scenario might be: Alice has 5 BTC on her side, and Bob has 1 BTC on his side.



(a) Initial scenario **(b)** Scenario after 10 transactions

At any time, either party can publish the current state on the blockchain. At that point, the balances on each side of the channel are allocated to their respective parties *on-chain*. In Lightning Network, this implies the closure of the channel; in Raiden Network, instead, it is also possible for a user to "withdraw tokens from a channel without closing it"[3] (having his/her counterparty's signature as well).

Transactions in the channel are not subject to fees except for the channel opening and closing ones. As a consequence, a substantial advance of LN is that it allows micropayments (see section 4.5: *Fees and micropayments*). In particular, since transactions in the channel have no fees (without considering payment routing, described in the following sections), users can be encouraged to send even small amounts of money. Thus, it is now possible to transact the smallest unit currently available, 0.00000001 BTC (one *satoshi*).

Another relevant feature of LN is the privacy it gives to users. Except for the first and last transactions, nobody outside the channel can see what happens inside it (see section 4.4: *Privacy*).

Multisignature addresses & HTLC

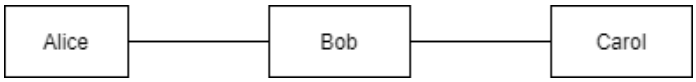
A multisignature (multisig) address is one that multiple private keys can spend from. When creating the channel, the number of private keys that can spend funds and how many of those keys are required to sign a transaction are specified. For instance, a 2-of-4 scheme would indicate that, on four possible keys, any two of them are required

[3]Quotation taken from "Withdraw tokens from a channel" section of `https://raiden-network-specification.readthedocs.io/en/latest/smart_contracts.html`

to authorize a transaction (signing it). It is substantial to point out that, when the multisig needs more than a single key to authorize a transaction, single users cannot move funds without other users agreeing.
A simple example: Alice (A) and Bob (B) lock up 5 BTC each into a 2-of-2 scheme. This scheme is composed of only two private keys capable of signing, and both are needed to move money. Supposing that A ends up with 9 BTC and Bob with 1 BTC, this scenario might lead Bob not to cooperate, locking his and Alice's funds in the multisig address (note that Alice needs Bob agreeing to unlock her funds). A mechanism that prevents the problem that might occur when one of the parties decides not to cooperate goes under the name of Hash TimeLock Contract (HTLC). In the *Security* section that follows shortly, the process by which Lightning Network prevents cheating is explained in detail.

Routing payments

A substantial part of the usefulness of LN is due to the connection between channels. The existence of different paths allows payments between users even if those are not directly connected. For instance, if Alice (A) opens a channel with Bob (B), and Bob already has one with Carol (C), Bob can act as intermediary by routing payments between A and C.



Intermediaries receive coins in the channel with the sender. Then, they effectively spend their funds in the route toward the receiver. A significant clarification is that the overall amount of money in a channel remains the same. The following representation extends the example above, with Bob routing a payment between Alice and Carol. The reported scenario shows a channel between Alice with a 5 BTC balance and Bob with a 2 BTC balance; the second channel, between Bob with 3 BTC and Carol with 1 BTC, is reported too. Supposing Alice wants to send 2 BTC to Carol, she should send 2 BTC to Bob; then, Bob, from the other channel, should send 2 BTC to Carol. The final scenario would be: Alice 3 BTC, Bob 4 BTC (Alice side), Bob 1 BTC (Carol side), Carol 3 BTC.

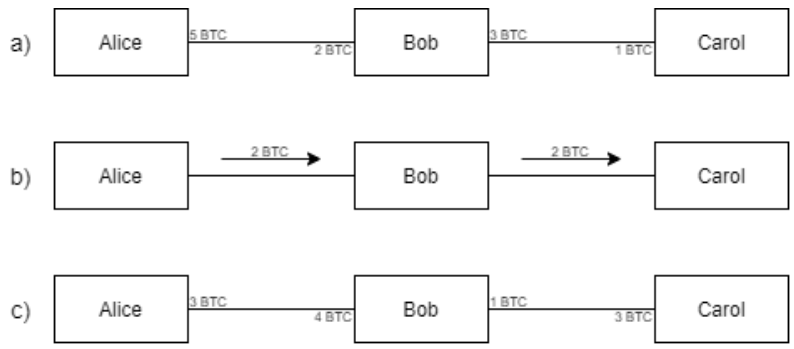


Figure: a) Initial scenario; b) Balance flows; c) Final scenario

The importance of routing payments in state channels Layer 2 solutions is exemplified by the fact that in Raiden Network, for instance, every payment, under the implementation point of view, is seen as a multi-hop transaction with N intermediaries; if the channel supporting it is direct, then N=0.

A criticism of routing payments is that it is impossible to spend more than the fund locked in a channel. Therefore, a considerable problem might occur when, considering the same example as before, Alice wants to send all her funds (5 BTC) to Carol. This transaction cannot be performed with Bob as an intermediary because Bob has just 3 BTC (Carol's side), which correspond to the maximum amount of money Alice can send to Carol via Bob.

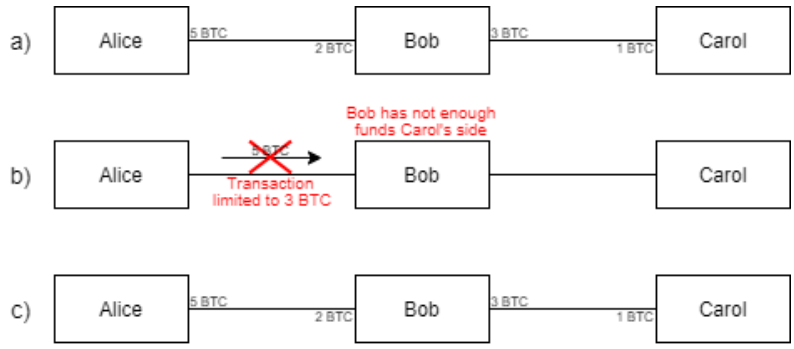


Figure: a) Initial scenario; b) Transaction denied; c) Final scenario as the initial state.

A relevant aspect to note is that, even if it is not mandatory, intermediaries usually ask for fees to route transactions. In particular, since they effectively spend their funds in involved channels, they ask senders to pay a fee (a small fee compared to that of the main chain) for having their payments routed to the recipient. This aspect might imply relevant consequences for the possibility of micropayments in a

not fully decentralized solution, as it is further detailed in *Decentralization* and *Fees and micropayments* sections.

4.1 Scalability

Layer 2 solutions aim at solving blockchain's inherent scalability problem. Lightning Network achieves this objective for Bitcoin's blockchain by increasing from almost 5 TPS to (theoretically) billions of transactions per day, with an estimated average of at least 11,000 TPS. It enables transactions to be confirmed instantly, securely (see section 4.2: Security), maintaining a decentralized protocol (see section 4.3: Decentralization), and anonymously (see section 4.4: Privacy), intrinsic characteristics of the blockchain. Although Lightning Network significantly increases the number of transactions per second, two scenarios, not convenient for using this Layer 2 solution, are worthy of consideration:

1. The first scenario involves payments to route in the network. In case even just a single node (within the routing path) has not enough funds in the channel, the requested payment will fail. This event would require the sender to seek another route towards the recipient, and the whole process may take longer than expected[4]

2. The second scenario which can slow down might occur in the case of uncooperative users. In particular, at the moment of channel closure, the presence of an uncooperative user (that voluntarily decides not to countersign the transaction) implies that the one who broadcasted the transaction has to wait for the timelock to expire before being allowed to spend the funds

An important clarification is that for the first scenario, a partial solution might be to divide large transactions into many small ones (*e.g.*, 1 BTC transaction might be split into ten smaller transactions of 0.1 BTC each). This paradigm will help to more likely find ways with the required amount of money. On the other hand, the second problem described has no remedy since it is a consequence of a high-security level granted by the Lightning Network.

4.2 Security

Hash TimeLock Contract is a mechanism to remedy any uncooperative behavior in payment channels. A HTLC consists of two parts: hash verification (based on

[4]Notice that a route leading to a payment failure for the reason described above cannot be "excluded" a priori and simply not proposed, because of privacy of the state channels (for this purpose, see section 4.4: Privacy)

hashlock mechanism) and time expiration verification (based on *timelock* mechanism). LN uses hash locks and timelocks to ensure payment security.

> A ***hashlock*** is a condition placed on a transaction dictating that funds can only be spent by proving to know a secret. The sender hashes a secret (any Byte combination can serve as a secret) and includes the hash in the locking script. The receiver can spend it only if he/she can provide the original data (the secret) that matches the hash. Note that the only way he/she can provide that data is if the sender gives it to him/her, but it is a very unrealistic situation: therefore, only the person who knows the secret that was hashed will be able to use the payment. A ***timelock*** is a condition that prevents from spending funds before a certain time. Timelocks require the production of a verifiable digital signature before a certain time.[5]

The idea behind HTLCs is that the receiver of a payment acknowledges receiving the transaction before a specific time by generating cryptographic proof of payment. After that time, the receiver cannot claim the payment anymore, returning it to the sender.
To better clarify the idea, suppose that Alice has to give 1 BTC to Bob; therefore, Bob will be the one to initiate the process with a payment request. The process can be summed up in three main points:

1. Bob generates a *payment_secret* and keeps it to himself
2. Bob produces the *payment_hash* by hashing the *payment_secret* and sends it to Alice
3. Alice creates a new commitment transaction that can be spent either by Bob if he can provide the *payment_secret* within a specific time (defined by the timelock) or by Alice if Bob has not provided the *payment_secret* by the timelock expiration.

For this mechanism to work correctly, the parties exchange the hashes of their secrets when the channel is opened. For each new transaction, the participants exchange the old secrets in plaintext and the hashes of the new ones.

HTLC allows overcoming the impasse problem described in one of the previous sections, in which a user decides to be uncooperative. At any moment, either participant can decide to sign and broadcast a transaction to the main chain. While the

[5]This paragraph is quoted, with just some modifications, from `https://academy.binance.com/en/articles/what-is-lightning-network`

counterparty can spend the funds immediately, the participant, who broadcasted the transaction, must wait for the timelock to expire to spend the funds.[6]

Here is a simple and concise example: considering again the 2-of-2 scheme with Alice and Bob, in which each participant put 5 BTC, we can assume that after a certain number of transactions, Alice has 9 BTC and Bob has 1 BTC on his side. Supposing Alice wants to redeem her funds, but Bob does not cooperate, she can anyway access her funds by broadcasting the last transaction to the main chain and waiting for the timelock.

A significant characteristic of the LN added by the use of HTLC is that it naturally *prevents cheating*. In fact, up to this point of the description, users can potentially broadcast an old transaction. However, secrets of the older transactions have been shared between the parties. Therefore, LN prevents these behaviors since the counterparty knows the secrets used for older transactions and can exploit them to get all the funds.

Despite everything, there is still a minimal possibility in which a malicious user can take advantage of the counterparty's offline period, broadcasting an old transaction without the other part being able to do anything within the timelock deadline (as he/she is offline). A significant clarification is that the vulnerability just explained has a very low percentage of occurrence, since the whole system is not managed directly by the parties but at a lower level of the network.

For what regards Raiden Network, instead, the last described problem can be mitigated: in fact, a node going offline can ask and rely on a Monitoring Service, which is basically a (set of) node(s) monitoring the channel and reacting to possible closures of the latter; in particular, the user pays a reward to the first node which discloses a cheating behaviour by his/her counterparty, while the user himself/herself is offline. Of course, the introduction of Monitoring Service solves one of the security problems of state channels L2, at the price of losing something in terms of privacy of Raiden Network.

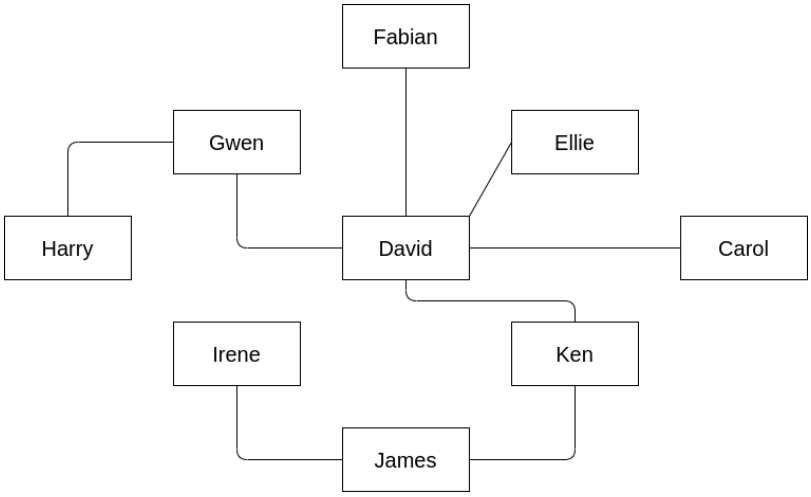
4.3 Decentralization

Given its *peer-to-peer* architecture, Lightning Network can certainly be considered as a decentralized framework, at least from a theoretical perspective. In fact, no entity or node manages or regulates *off-chain* transactions, which are directly performed over the *peer* network in a single-hop or multi-hop way.

However, even if decentralization is fully guaranteed under the theoretical point of view, the real structure of the network actually has scale-free properties: the majority

[6]Note that this problem arises only in an uncooperative case: if both participants sign the transaction, they can both spend immediately.

of LN nodes has active channels (of limited capacity) with few *peers*, while a limited number of *hubs* is connected to a high number of nodes. It is interesting to understand that this *hub and spoke* architecture is quite likely to be "fed" by single users' decisions: suppose, for instance, that Alice (A) and Bob (B) enter Lightning Network in order to establish a payment channel; their idea is that of starting transacting between themselves, but also becoming active users of LN by later exchanging cryptocurrencies with other users, too. The structure of LN, for this example, is imagined to be the following:



Alice and Bob have basically two choices: creating a direct channel between themselves or creating an indirect one. Of course, a direct channel would ensure no fees for all the transactions between A and B, except for the fees payed on the main chain; on the other hand, if any of the two also liked to transact with Carol (C), Gwen (G) and James (J), he/she would necessarily have to open other channels. The indirect channel, instead, would add LN fees for transactions between Alice and Bob, but would require no other channel for transacting with C, G, J.

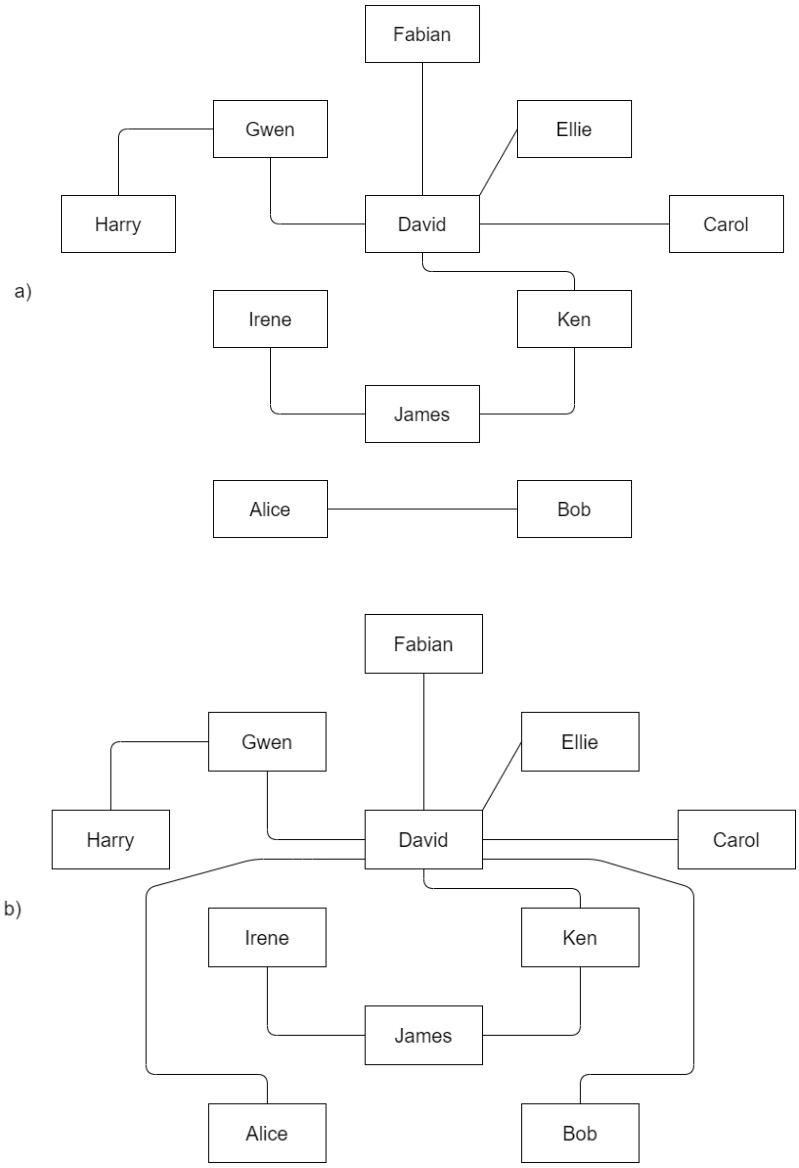


Figure: LN topology after the creation of: a) direct channel between Alice and Bob; b) indirect channel between Alice and Bob, with *hub* David as intermediate node

Since *off-chain* fees are sensibly lower than main chain fees, and given the fact that many nodes are likely to enter LN and keep channels active in order to transact with more than one other *peer*, most of them decide to open a state channel with a *hub* (instead of multiple ones with "target" nodes); in the previous example, at least one between Alice and Bob (if not both) are likely to open their channel with David. In reality, the presented concept is just a general idea, because also channels' capacities

play a key role in the evolution of LN architecture; in any case, several studies have shown that scale-free properties have been emerging in Lightning Network, and that users tend to prefer setting channels with reliable and well-established central nodes, so to pay the smallest amount of fees (few hops with approximately every other *peer*, without the necessity to open various channels, which has a considerable *on-chain* cost) and be connected with the majority of nodes.

So, Lightning Network certainly has the idea of preserving decentralization as its core: in fact, it is implemented as a *peer-to-peer* network. However, the real structure of this L2 solution tends to be less distributed than one might think. This aspect can not be considered an unbalanced step towards centralization of the blockchain, since, in any case, the architecture is still P2P and final summaries of transactions are reported *on-chain*. However, it is also important to take into consideration the real structure of LN, in order to better understand possible issues for robustness (in a scale-free network, if one or more *hubs* are offline there might be problems, since they constitute the "core" for connectivity), fee policy (as explained later, in section 4.5: Fees and micropayments) and privacy (see section 4.4: Privacy). Moreover, another aspect should be underlined: *hubs* are not only central nodes, but tend to be also the only nodes which can route substantially high-value payments (in fact, most of users have only low capacity active channels); this fact is not a huge issue since LN is mainly a network of micropayments (low fees make it possible to spread a transaction into a set of smaller ones), but it is certainly another considerable part in the overall analysis.

4.4 Privacy

Privacy of state channels Layer 2 solutions, such as Lightning Network, has to be evaluated under two different perspectives: the first one is related to confidentiality about transactions between two nodes, as well as their balances in active channels in which they are involved; the second, instead, is strictly connected to routing, which is an essential part of the technology.

LN offers to users the possibility to transact for an indefinite number of times, by reporting on the blockchain only an opening and a final transaction: this implies a high level of privacy between transacting nodes, since the only information which is made public is the balance of the channel at "time 0" and at closing time. In practice, under this point of view, state channels guarantee more privacy than the main chain.

In order to make routing possible, routing protocol over Lightning Network shares IP addresses of nodes and capacities of active channels: it is useful to remember that the capacity of a channel is the (fixed) amount of cryptocurrencies locked in the multisignature address, while balances summarize the (dynamic) evolution of funds between the two nodes. The most important information to be hidden is certainly the

balance of a channel; in fact, nodes involved in an active channel and the capacity of the latter are on the blockchain since the setting of the channel itself. Hiding balances is Lightning Network's key feature for confidentiality of state channels, although it creates a bit of inefficiency in routing payments: in fact, given the impossibility to know whether all edges in a route have enough funds to support the transaction, more than one attempt might be necessary to correctly finalize a payment.

For what regards multi-hop payments, another aspect ensuring confidentiality is an *onion-routing* protocol, for which the sender creates several cryptographic layers which allow every intermediate node to be aware only of the identity of the predecessor and the successor in the path.

In order to summarize, transactions on direct channels are completely private, while transactions with intermediaries are quite confidential; in the latter case, the longer the routing path, the higher the privacy degree. It is important to notice that a transaction involving only one intermediate node is completely not private in a certain sense, because the routing node is fully aware of the payment's size and direction; however, given *onion-routing* approach, this node does not know whether the payment starts from its predecessor and is destined to its direct successor, or it is a longer-path transaction. This consideration is certainly valid, but it can be in some sense threatened by the scale-free topology of LN; in particular, most of multi-hop payments are directly routed across a limited number of *hubs*, so the general tendency is to have short paths for routing transactions (and short paths certainly mean a lower level of privacy with respect to longer ones).

Anyway, privacy guaranteed by Lightning Network is quite robust, especially for transactions on direct channels or long routing paths. Some issues may arise, of course, but they are caused by the topology of the network, not by the "privacy protocol" itself.

For what regards Raiden Network, instead, it is worthy repeating that a bit of privacy is "sacrificed" in order to implement a Monitoring Service, which guarantees a higher degree of security (see section 4.2: Security).

4.5 Fees and micropayments

As anticipated in the previous sections, Lightning Network transactions in a direct channel are not subject to fees, while intermediate nodes that route indirect payments generally ask for them (even if they are not mandatory). The main reason why LN fees are null or sensibly lower with respect to *on-chain* fees is that almost negligible computational effort is necessary to process a transaction on state channels; small fees present in the network, in fact, are simply related to the fact that, for acting as an intermediate router, a node should unbalance two of its channels: in order to accept this fact, an incentive is asked.

Since LN fees exist only in multi-hop payments, and routing paths depend on the topology of the network, the fee market has been evolving in accordance with Lightning Network's structure. Up to the moment of writing, the evolution of fees is still in an early phase and, as happened to many blockchains in their first times, the additional cost for transacting is very low. However, even if there is no reason connected to computational effort or scalability that forces fee price to rise, Lightning Network's free-scale topology might lead to an evolution.

LN fee market is driven by two main sources:

- Every node acting as an intermediate router sets the fees it is going to apply: typically, fees are composed of a variable part (proportional to the size of the payment to be routed, since the higher the payment, the bigger the unbalancing factor for intermediate node) and a fixed part

- Senders of multi-hop payments can decide which routing path to follow

In practice, senders are likely to prefer routing paths that charge the lowest fees; these paths are usually among the shortest available to reach the recipients, since every intermediate node may ask for fees. In some sense, short paths are better than long paths for what regards fees, while they are worse in terms of privacy.

The important aspect to consider is that, in the case in which Lightning Network *hubs* significantly centralize routing payments, they could decide to charge higher fees, and this can be a problem for micropayments. In fact, suppose Alice is not directly connected to Bob in LN, and needs to send a single micropayment to him. Alice's options are either transacting *on-chain*, or opening a LN channel with Bob, or routing a payment to him: of course, neither the first nor the second possibility are convenient, since they would involve at least one transaction on the main chain, with related fees; so, the only option for Alice is to route the transaction. Imagine now that Lightning Network topology is quite centralized, and at least one *hub* should be traversed in order for the payment to be routed: in this situation, since Alice's only option is transacting *off-chain* without opening new channels, she is going to accept not only very low fees (as the ones present in LN at the moment of writing), but a bit higher ones. This example shows that, dependently of the topological evolution of the network, fee market may evolve in one direction or another. Up to most recent data, fees in LN are low, and this is completely in accordance with the idea of this L2 solution: offering state channels to transact, which imply no additional cost under the theoretical perspective. As for privacy's situation, also for fees the most important source of issues in the future may arise from the free-scale structure of the network.

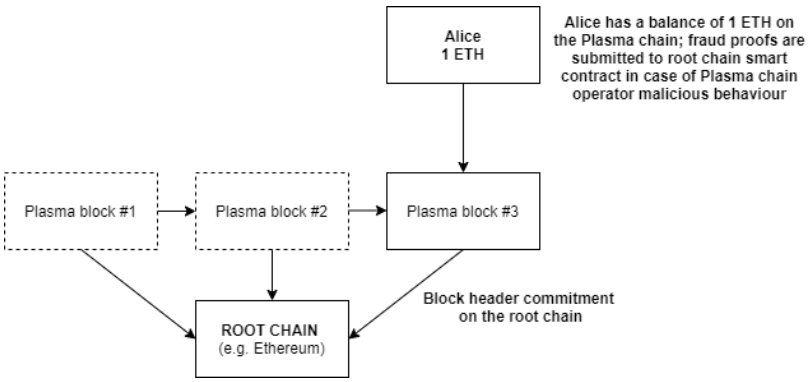
5 Plasma

Plasma is a Layer 2 blockchain scalability solution for Ethereum proposed by Joseph Poon and Vitalik Buterin in 2017. Plasma aims at extending the concept of sidechains, as a way to reduce the number of transactions to be processed by the L1 blockchain, also going to reduce transaction latency and cost.

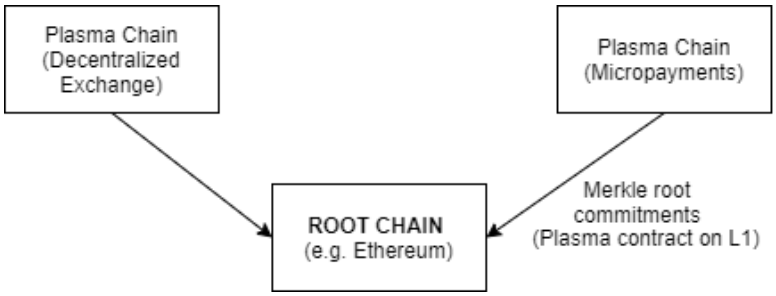
Standard sidechains allow the deposit of assets on a contract located in the L1 chain (e.g. a smart contract on the Ethereum blockchain), which will be monitored by the sidechain operator, to be able to credit the assets themselves to users in the sidechain. Greater scalability is achieved thanks to the different consensus mechanisms, such as, for example, proof-of-authority, which allows for faster block times and thus a dramatic increase in transaction throughput, with very low fees (economic incentives might mainly derive from the service the operator is willing to offer).

However, as stated in the well-known scalability trilemma, these advantages come at a cost in terms of centralization and lack of security: as an example, in a proof-of-authority sidechain, the set of validators, which are allowed to generate new blocks, could stop producing new blocks and stop processing exit requests from the sidechain (i.e. not allowing anyone to withdraw funds previously deposited in the sidechain itself). In other words, standard sidechains imply the need for trust in the chain operator.

Plasma proposal's goal is to solve this problem by publishing each sidechain's block header to the L1 chain (the root chain), therefore minimizing trust but at the same time allowing verifiable fraud proofs and enforcible state, given that the root chain is secure and constantly available. This could lead to move a huge amount of load in terms of transactions, from the root chain (L1) to Plasma chains (L2), with only periodic commitments on the root chain. Moreover, Plasma reduces the storage needed in the root chain, by compacting several state transitions in a single merkle root commitment.



Ideally, multiple Plasma chains can be created, each one with a different purpose and use case. As an example, one could create a Plasma chain to handle transfers involved in a decentralized exchange application, while another Plasma chain could be used to handle micropayments. In addition to this, child chains could be organized as a tree of chains, in a court system fashion, where the chain at height one of the tree can solve disputes arising from any height two chain, as it holds all blocks headers and can evaluate proof of frauds. The proposed design resembles the flow of the *MapReduce* computation paradigm, by assigning workloads to child chains (at various depth of the tree), which then commit work up to the root chain.

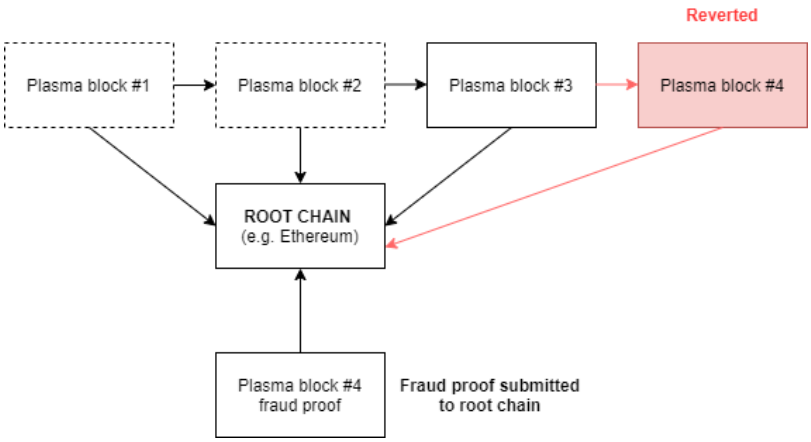


The described construction allows user to actively participate only if their *on-chain* balances are being updated or some malicious behaviour on the Plasma chain is being detected. Funds can be held in the child chain, and they are secured by a full representation on the root chain: assets can then be withdrawn after a dispute mediation period, where others can submit fraud proofs. Fraud proofs can stop a withdrawal process if irregularities are detected, for example if the coins are proven to have already been spent. Penalizations are put in place, by appending a bounty alongside withdraws and proof of frauds which is lost in case of faulty behaviour.

To achieve this verification scheme, an UTXO (Unspent Transaction Output) model is adopted for the Plasma chain (at least in the initial proposal), because it allows more compact ways to verify if a particular state has been spent, and, as a result, it is more efficient for fraud proofs and withdrawals. This idea is in contrast with Ethereum account-based state model: in UTXO chains there is no concept of balance, but there is a set, a data structure that is updated every time a new transaction is sent. Every transaction consumes an output in the previous set, which was fed by a previous input; the new output produced becomes an input for the next generated element of the set. In this type of chain, the final balance for a given address is reconstructed by summing all the unspent outputs for that address. The UTXO model exploited by most Plasma chain implementations causes general computations (e.g. smart contracts execution) not to be supported; only simple transactions such as transfers or swaps are possible on the majority of Plasma variants.

Fraud Proofs

Plasma child chains are operated by validators, which propose blocks. Plasma's enforcible state restricts their possible malicious behaviour by making use of fraud proofs; in case a malicious block is propagated, any other actor that is monitoring the child chain and receives the block can submit a merkleized fraud proof on the parent blockchain. The invalid block is then rolled back, and the proposer of the faulty block is penalized.[7]



Both fraud proofs and withdrawals are secured by the concept of merkle proof. The information that is committed by the Plasma chain to the root/parent chain is the merkle root of the block that has just been generated. Whenever a user wants to perform a withdrawal, a request is submitted to the root chain contract, together with a merkle proof, that is checked by the contract: these checks consist of making sure that the transaction which generated the output that the user is willing to withdraw was included in a certain block in the past; moreover, the contract checks if the specific output *belongs* to the user who is performing the request.

However, the user cannot prove that that output has not already been spent within the Plasma chain; therefore, a challenge period is needed in order to allow other actors to possibly provide proofs (still in the form of merkle proofs) showing that the output cannot be redeemed.

For this mechanism to work, block data availability is needed, in order to be able to produce fraud proofs. It is important to note that, after some time and if the parent blockchain reaches sufficient finality, commitments are considered finalized and cannot be reordered.

[7] All images in this section have been inspired by `https://plasma.io/plasma-deprecated.pdf`

Deposits and Withdrawals

Plasma's *on-chain* architecture is composed of contracts that have the following responsibilities:

- Keeping track of block header hashes (merkle roots) commitments sent by Plasma child chains
- Processing submitted fraud proofs and withdrawals, and handling the related penalization for the invalid ones

Deposits are initiated by the user by sending funds to the contract located in the root chain. The Plasma chain detects that a deposit has been started, and includes in a block a commitment to the fact that funds will be spendable. Finally, the depositor signs a transaction on the Plasma chain, to declare that he/she has seen the commitment made by the chain in a previous block.

The withdrawal process is composed as follows:

1. The user sends a signed withdrawal transaction on the root/parent chain. The withdrawn amount must correspond to the whole unspent output. An additional bond amount is put at stake to allow for the penalization of invalid withdrawal attempts
2. There is a predefined timeout period to allow for disputes. In case valid proof is submitted, the withdrawal is canceled and the bond staked before is lost (this is useful to discourage faulty withdrawal attempts). If no fraud proofs are found, then the user is allowed to redeem the requested funds

Withdrawals as described above are quite slow in nature, due to the dispute period needed to allow others to provide fraud proofs. Usually, a period of time considered adequate as challenge period is 7 days. This mechanism protects users from frauds, but also comes at a great cost in terms of usability and user experience.

Fast withdrawals are possible too, exploiting liquidity providers (LP) willing to offer this as a service. If the liquidity provider is able to fully validate the Plasma blockchain, then user funds could be locked to a contract on the child chain. The behaviour enforced by the contract is that, if a certain transaction proof stating that the LP has sent funds to the user can be found on the root blockchain and has been finalized, then the payment to the LP will go through in the Plasma chain; otherwise, the sender can redeem back his funds. Liquidity providers are incentivized to provide this service by charging extra fees that the user has accepted to pay in order to have a faster way to exit the Plasma chain (with respect to a simple withdrawal).

Basically, this mechanism implements an atomic cross-chain swap, and can also be exploited to move funds between Plasma chains.

5.1 Scalability

In terms of scalability, a Plasma chain can claim the same performance improvements achieved by standard sidechains. Obviously, the transactions throughput and latency that can be obtained ultimately depends on the consensus mechanism that the Plasma chain exploits. Usually, we can expect child chains to use a consensus algorithm other than proof-of-work, such as proof-of-stake (PoS) or proof-of-authority (PoA), which allow for faster block times and a higher number of transactions per second (TPS).

As an example, *Polygon* (formerly known as *Matic Network*), a commercial adapted implementation of Plasma exploiting PoS as consensus mechanism for the Plasma sidechain, can be considered. Polygon claims to have reached a peek of 7,200 TPS on its internal testnet; at the moment of writing, Polygon mainnet has an average block time of 2.1 seconds and a gas limit per block of 20 million. Using current available data, simple statistics can be computed, in order to get the current average gas consumed for each Ethereum mainnet transaction:

$$Txs \approx 1,500,000 \; transactions \; per \; day, \;\; Blocks \approx 6,500 \; blocks \; per \; day$$

$$Average \; transactions \; per \; block = \frac{Txs}{Blocks} \approx 230 \; transactions$$

Considering the current Ethereum mainnet block gas limit (12,500,000):

$$Average \; gas \; per \; transaction \approx \frac{12,500,000}{230} \approx 54,350$$

As a consequence we can obtain an approximate value for the current Polygon throughput:

$$Throughput \approx \frac{20,000,000}{Average \; gas \; per \; transaction} \cdot \frac{1}{2.1 \; seconds} \approx 175 \; TPS$$

The obtained result is a good improvement (about 11 times) over Ethereum mainnet current thoughput, but still far from what is needed in order to envision mass adoption of the technology and handle, for example, the load implied by global retail transactions. However, it is interesting to notice that the approximation adopted here is quite conservative: blockchains with similar consensus mechanisms (not necessarily implementing Plasma) exploit a greater block gas limit and, potentially, a lower block time: this could lead to further improvements in terms of throughput. Moreover, this type of construction implies that other additional Plasma sidechains can always be added, therefore enabling *horizontal scaling* of blockchains.

5.2 Security

Security on a Plasma chain is ultimately achieved by merkle proofs submitted to the root chain, which prevent child chain's validator Byzantine behaviours and invalid attempts to withdraw outputs (such as those already spent).

Most of the security concerns in this context are related to data availability and block withholding attacks (BWA), which happen when validators refuse to publish new blocks. Users of a Plasma chain must constantly monitor the chain and, in case of BWA, they need to perform a *mass exit*. In addition to the fact that the need to constantly monitor the child chain (*liveness assumption*) creates usability issues, one of the main challenges in Plasma is to make mass exits compact and efficient. Mass exits cannot be performed as *fast withdrawals* since, in this case, the chain is assumed to be Byzantine and liquidity providers would not have the interest in process this kind of withdrawals. Even if it is theoretically possible to use the simple withdrawal process described in the previous section, a dedicated procedure has to be designed to mitigate data availability issues as much as possible.

The mass withdrawal interactive game proposed by Buterin and Poon is based on a mass exit operator, which verifies the Plasma chain up to the point in which the BWA started. An exit transaction is created with an attached massive bond and users sign of the mass withdrawal. The operator will then sign and broadcast the *mass exit initiation transaction* (MEIT) to the destination chain, optionally charging a fee to every user who is participating in the process. The operator also publishes a full *bitmap* of the state involved in the exit: this let observers of the Plasma chain to understand what is being withdrawn and optionally challenge it; finalization of the MEIT may take a lot of time, in the order of weeks.

However, the described design leads to problems that are intrinsic to mass-exits: supposing a very popular Plasma chain with a very big number of users, a fraudulent event by validators could cause the root chain to be flooded by huge simultaneous mass-exit transactions. As a result, the L1 chain would likely be heavily congested.

Other possible security threats may include:

- Smart Contract code vulnerabilities: smart contracts on the root chain are responsible to enforce the state of the child Plasma chains. The security of the Plasma architecture is entirely reliant on complex *on-chain* smart contracts; the complexity needed in this context increases risks of vulnerabilities
- Redeem transactions may be too expensive on the root chain, thus making withdrawals unfeasible
- State enforcement and execution of exit transactions might fail if a 51% attack is being performed on the root chain and blocks are being censored

5.3 Decentralization

From a strictly theoretical point of view, the level of decentralization offered by a Plasma chain exclusively depends on the exploited consensus mechanism of the chain itself. However, Plasma is by construction trying to let not fully decentralized sidechains (such as those using proof-of-authority) to avoid the usual problems deriving from a centralized chain, through the use of merkle proof commitments on the root chain. In practice, the effects of the decentralization level chosen for a Plasma chain depend on two factors:

- As previously stated, the consensus mechanisms of the Plasma chain
- *When* transactions are considered finalized

To avoid side effects deriving from an increased level of centralization, one should consider a transaction finalized only when the merkle root of the block containing the transaction itself is committed to the root chain: doing this ensures that the state is being enforced by fraud proofs, and that the security is the same as L1 blockchain. To mitigate the need for waiting the commitment, the Plasma chain could make use of a proof-of-stake consensus mechanism, in order to find a balance between decentralization and achieved scalability: for this reason, *Plasma Proof of Stake* was proposed by Buterin and Poon, also trying to reduce the risk of block withholding attacks.

5.4 Privacy

Although privacy issues are not a primary concern in Plasma design, some considerations regarding the consequences of the proposed solution can be done. Essentially, Plasma is based on the concept of avoiding broadcasting all transactions in the root ledger. The load on the L1 chain is reduced, and the resulting privacy is determined by the specific child chain design, which could also be focused on guaranteeing greater privacy properties with respect to public ledgers like Ethereum or Bitcoin. Theoretically, implementations for such a Plasma chain could be inspired by already existing designs which aim to provide an increased level of privacy, such as *ZCash*. A solution of this kind would widen the scope of this Layer 2 solution, allowing not only for scalable public blockchains, but also to add privacy features to already existing public decentralized ledgers.

5.5 Fees and micropayments

In short, fees on the root chain are useful to increase security and avoid spam attacks, and to serve as an incentive for validators or miners to behave correctly, especially

in proof-of-work blockchains.

In Plasma sidechains, transactions fees can be reduced dramatically. Disincentives for extremely low value transactions and spam attacks are still a requirement, but there is no more the need for a particularly decentralized consensus algorithm, since users can always exit and return to the secure and decentralized root chain. As a consequence, a child chain can be designed, for instance, with a proof-of-authority consensus; this fact means that there are no longer miners interested in the "race" for high fees (situation in which miners prioritize transactions with higher fees, causing many blockchain's use cases to become unfeasible due to transactions costs). So, it is clear that Layer 2 solutions like Plasma enable features for blockchains that, in practice, are not possible on current implementations of L1 chains.

As an instance, *micropayments* are one the new possibilities enabled by Layer 2 scaling. One possible design choice is to create a Plasma chain specific to offer micropayments. A solution like this would allow real world scenarios like coffee shops using Plasma to receive payments by customers: the shop would need to monitor the child chain to detect malicious validators behaviour and periodically pay a fee on the root chain to withdraw earnings, for example once a week. Micropayments are considered a well-suited Plasma use case also for the resource saving in terms of storage which the root chain could benefit from (we assume that very low value transfers do not need the level of security offered by L1 chains), as a great number of transactions can be compressed in a light block header. In addition, another factor that favors this kind of low value transfers is the fact that, in a Plasma chain, it is possible to send assets to users who are not currently in the set of participants, unlike classical state channels solutions.

Plasma variants

Since 2017 many different designs, derived from the above described Plasma specification, have been proposed. The most significant ones are briefly described below.

- Plasma MVP: Minimum Viable Plasma is a proposed simple UTXO-based Plasma chain. It targets high volumes of payments, but does not support general computation (smart contracts). MVP's consensus mechanisms is proof-of-authority, thus it can also be used for private blockchains.

- Plasma Cash: user funds deposited to the Plasma chain are represented by non-fungible tokens (NFTs). Consensus is typically implemented as proof-of-authority or proof-of-stake. Thanks to the fact that Plasma Cash makes use of *sparse merkle trees* instead of standard ones, the solution is highly scalable, but suffers handling applications that need to deal with fraction of assets (an

example include micropayments). As a result, this variant seems to be appropriate to directly handle NFTs *off-chain*, which is an extremely useful feature for supply chains or card games applications.

- Plasma Debit: it aims to mitigate problems deriving from Plasma Cash by making use of payment channels inside the Plasma Chain, with operators as *routers* between users involved in the transfer. Therefore, any user must have payment channels set up with the chain operators. By using channels, this solution allows for transfers of fractional parts of funds.

These new versions of Plasma would undoubtedly deserve a detailed discussion, which (however) is outside the scope of this document.

6 Rollups

Payment channels and Plasma are usually considered "*full*" Layer 2 solutions, while *Rollups*, a framework for Ethereum proposed in 2018, is considered a "*hybrid*" solution between L1 and L2 scaling: this is due to the fact that, while Lightning Network and Plasma summarize many state transitions in one single commitment or channel closing, the Rollup proposal implies that some information on every single transaction sent on the L2 is posted *on-chain*. The idea of Rollup is that transactions are aggregated *off-chain*, causing a reduction in congestion and fees on the root chain.

The concept of Rollups is somehow similar to the one previosly explained for Plasma: a smart contract deployed on the root chain keeps track of the current (most recent) merkle root of the state of the *rollup* (in practice, this could be the L2 chain). The computation of the new state is performed *off-chain*, and the root chain is used for data availability (by posting a piece of data for each transaction). The described design allows to avoid data withholding issues, because all the information is always retrievable from the root chain, which is considered to be secure and always available. Any actor is able to publish a *batch*, which is a collection of compressed transactions with the previous merkle root and the new merkle root attached to it. At this point, the Rollup contract, before updating the state root with the new one, has to check (for security reasons) if the previous root corresponds to the current root (for a focus in the security analysis of the solution, refer to next discussions in the section). A peculiar feature of Rollup is that it allows to transact outside the Rollup contract itself: this is meant to support transactions whose input comes from outside or whose output is destined for outside. At the time of writing, there are mainly two types of Rollup solutions: ZK Rollups and Optimistic Rollups. They differ in how they perform verification on the submitted batch.

ZK Rollups

The ZK Rollups solution is based on the concept of *validity proof* and *zero-knowledge proof*. The idea behind ZK Rollup is to bundle every batch with proof of their validity. The proof should be easy to check, proving the correctness of the batch content. ZK Rollup framework uses the SNARK proof (see section 6.2: *Security*): this kind of system allows observers to immediately prove the validity of an assertion. So, the main characteristic of this proof is that it is cheap to verify. However, computing it is expensive. Therefore, ZK Rollup is an appropriate system for transactions' management, but it does not fit with complex contracts execution.

Optimistic Rollups

Optimistic Rollups is an *interactive* approach. It is, in a sense, more similar to Plasma's, since it involves the use of *fraud proofs* (see *Fraud Proofs* in Section 5). In this kind of solution, new batches (and, therefore, new merkle roots) are published by operators, without being proved to be right by the Rollup smart contract. This mechanism is further detailed in section 6.2: *Security*.

6.1 Scalability

One of the key points of Rollups scaling is compression: in practice, every Rollup transaction (remember that information of every transaction is reported *on-chain*) may be compressed to occupy a total space of (down to) approximately 12 Bytes. This size, of course, varies with Rollup and transfer types: Optimistic Rollup transactions need more Bytes (information) for later verification (due to the absence of a SNARK proof); ETH transfers take less Bytes than more "complicated" transfers, as it normally happens on Ethereum blockchain. So, 12 Bytes lower-bound for a Rollup transaction refers to an ETH transfer performed using ZK Rollups.

To better understand the translation of this compression feature into throughput achievements, a best-case scenario can be considered: ZK Rollups for transferring ETH. Considering the current values, on Ethereum blockchain, of:

- Gas limit: 12.5 Million gas/block
- Gas per Byte in L1 transaction: 16 gas/Byte
- Average block time: 13 sec/block

Supposing to spend 1 Million gas for proof verification, the following calculations can be developed:

$$Block\ size = \frac{Gas\ limit - Gas\ for\ proof}{Gas\ per\ Byte} = \frac{(12.5 - 1)\ Million\ gas/block}{16\ gas/Byte} \approx 715,000\ Byte/block$$

$$Transactions\ per\ block = \frac{Block\ size}{Rollup\ transaction\ size} \approx \frac{715,000\ Byte/block}{12\ Byte/tx} \approx 59,500\ tx/block$$

$$Throughput = \frac{Transactions\ per\ block}{Average\ block\ time} \approx \frac{59,500\ tx/block}{13\ sec/block} \approx 4,500\ TPS$$

It is crucial to remember that this is a theoretical achievement for ETH transfer on ZK Rollups, considering that the whole *on-chain* transaction is composed of data coming from the Rollup itself. In any case, although further complications may be analyzed, 100x scalability improvement is a good approximate evaluation of ZK Rollups scalability with respect to Ethereum L1 throughput.

For what regards Optimistic Rollups, instead, considering a Rollup transaction size of 12 Bytes is not possible anymore, since further data must be committed on the main chain for verification; however, even considering a transaction size 6 times bigger than the one for ZK Rollups (which is a quite reasonable value), the obtained throughput is approximately more than 800 TPS: so, still substantially higher than Ethereum current throughput for token transfers. At the time of writing, one of the main features offered by Optimistic Rollups with respect to ZK Rollups is *general computability* (of smart contracts). In any case, also for complex contract interactions the improvement measured as a ratio with respect to L1 scalability is reasonably the same as ETH transfers, since other contracts normally require more computation on the main chain too.

In addition to pure throughput analysis, it is important to analyze the withdrawal time: in fact, users' purpose is not only to transact at high speed, but to drop out fast too. At this purpose, the two Rollup solutions behave very differently, due to security verification diversity (see section 6.2: *Security*): while for ZK Rollups the withdrawal period is very fast (thanks to SNARK proof, waiting for the next batch is sufficient), in Optimistic Rollups this time goes up to 1 or 2 weeks, so to be compatible with fraud proof challenge period.

6.2 Security

Dealing with security in the Rollup system, it is crucial to divide the discussion by distinguishing ZK Rollup and Optimistic Rollup.

ZK Rollup is a schema that involves two kinds of users: transactors and relayers. The formers are users who are willing to perform transfers; relayers, instead, are users that have staked a bond in the Rollup smart contract and are in charge of collecting a large number of transactions to create a *rollup*. ZK Rollup, as introduced in a

previous section, exploits the SNARK proof system.[8] Relayers (that be assimilated to operators of a Plasma child chain) are in charge of generating the SNARK proof, which is a hash that represents the delta, the difference between the precedent state of the blockchain and the new state, after the execution of the transactions that compose the *rollup*.
After transfers have been bundled up, the batch is sent to the root chain contract together with the SNARK proof, guaranteeing that the new state has been correctly generated by the transactions included in the batch itself (starting from the old merkle root up to the new one). This kind of approach can be considered as an *explicit verification*, which eliminates the need for fraud proof. Since their verification complexity is $\mathcal{O}(1)$, SNARKs are well-suited for transaction managing and small contract execution applications. However, as already anticipated, this mechanism does not fit well with more complex contracts' execution due to their complicated proof generation.

The Optimistic Rollup approach, instead, is similar to Plasma from a security point of view. Both of them implement the same proof mechanism: fraud-proof. This interactive approach allows overcoming the complex contracts execution management. The general idea is that the *rollup* contract keeps track of the history of updated states, and anyone (challenger) who detects a wrong post-state root can publish proof of that incorrectness. At that point, the contract can verify the provided proof: if the proof points out an invalid submitted assertion, the system restores the state considering the last valid batch and penalizes the publisher.

6.3 Decentralization

In addition to what already presented in the discussion dedicated to Plasma decentralization (see section 5.3), which can be partially extended to Rollups, a few points should be considered.

First of all, it is important to remember that Rollups are implemented *on-chain* through a smart contract: this, of course, implies a higher rate of centralization, especially if Rollups usage becomes substantial and, in consequence of this, also the volumes managed by a single smart contract. On the other side, contrary to other kinds of L2 solutions (such as state channels), in Rollups all data needed to reconstruct every state is published *on-chain*, thus ensuring a consistent level of decentralization under this point of view.

An important centralization issue concerning ZK Rollups, however, is the following: since SNARK proofs must start from a trusted initial condition, the latter can

[8] A detailed description of SNARK proofs goes outside the goals of this work; the interested reader can find a meaningful reference at `https://z.cash/technology/zksnarks/`

be a source of centralization for the early life of the Rollup, because it is controlled by a small group. In any case, the problems related to this aspect are closer to possible attacks mining security rather than menaces for the centralization of the L2 solution.

6.4 Privacy

Rollups cannot certainly be considered as a Layer 2 solution which has, among its biggest interests, privacy; in particular, by reporting on the blockchain a "summary" of every single transaction performed *off-chain*, the privacy level is certainly lower than all the solutions committing only an initial and a final transaction. One countermeasure to this can be found, at least for Optimistic Rollups: they can act as Layer 3 scalability solution on top of an already existing L2 for Ethereum ensuring privacy of transactions; of course, this proposal comes at the cost of excessive fragmentation, which translates the "privacy countermeasure" for Rollups into a not sufficiently good idea.

At the moment of writing, privacy concerns for Rollups are not considered as a primary deal, and the community is further studying and developing solutions for higher throughput, increased security and low fees much more than focusing on privacy.

6.5 Fees and micropayments

Fees in Rollups are composed of two parts: *off-chain* and *on-chain* ones. Of course, the cost associated to every transaction depends also on the type of used Rollup solution: as explained in previous sections, Optimistic and ZK Rollups differ for transaction size and verification mechanism.

In order to present a brief example[9] and give the flavour of the effects on fees given by using Rollups, a simple use-case can be highlighted: ZK Rollups used for cryptocurrency (ETH and ERC-20 tokens) transfers. For this purpose, the last ten batches[10] (at the moment of writing) by *zkSync*, an implementation of ZK Rollups, are taken into consideration. By analyzing the blocks committed and verified on the Ethereum mainnet, it turns out that each of them contains data related to between 129 and 206 transactions; the *on-chain* fees associated to a block range from 0.052 Ether to 0.076 Ether, with consequent average *on-chain* cost per transaction

[9] All data considered in the presented example comes from `https://etherscan.io` and `https://zkscan.io/`

[10] The decision of considering only ten batches directly derives from the scope of the example itself: providing some approximated numerical estimations as a comparison between Rollups fees and main chain ones, and not a full analysis.

of 0.000461 ETH (∼1.12 USD, at the moment of writing). These *on-chain* fees are paid by validators, when a block is published on L1. However, in this case we are considering as all transactions in the blocks were transfers, but, in general, different type of *off-chain* operations have different costs (e.g. withdraws are more expensive than transfers); therefore, the estimated fee must be considered as an *upper bound* value for transfer fees. As a matter of fact, considering the last ten *zkSync* transfers, an average fee of 0.000046841 ETH (currently ∼0.11 USD) was observed. It is worth to notice that this fee cost diverges from that estimated by the *zkSync* team (around 0.001 USD). For a more detailed discussion, please refer to the section 8.3.2: proof of concept usability.

The presented example finds its conclusions when comparing Rollups fees with Ethereum fees: for this reason, the last ten transactions (at the moment of writing) concerning transfers on the mainnet are analyzed, discovering that their average fee cost is approximately 1.23 USD. In practice, Rollups fees for cryptocurrency transactions are, in the analyzed scenario and data, up to 11 times cheaper than those performed with no L2 solution.

Of course, one of the goals of Rollups is to decrease transaction cost more than what was inspected by the presented example. The actual values, in fact, are driven by the same reasons for which also the scalability cannot reach its theoretical achievement. In any case, a massive introduction of Layer 2 solutions would help fulfilling this goal.

Moving to micropayments, the concept is the following: 0.11 USD (the average computed value considering current available data) still seems to be quite high considering, for instance, a transfer to pay a coffee (1 USD). This line of reasoning is true for the current situation; however, Rollups have the potential to decrease current Ethereum fees by orders of magnitude, thus enabling the possibility of payments of whatever scale.

7 Comparative table

In this section, a comparative table of the analyzed solutions is presented. The table aims at bringing out the key points of each framework, by highlighting strengths and weaknesses for each of the five aspects under consideration. For a detailed content on individual solutions, please refer to section 4 (for Lightning Network), section 5 (for Plasma) and section 6 (for Rollups).

	LN	**Plasma**	**Rollups**
Scalability	It increases scalability by theoretically reaching around 10,000 TPS. However, there are some scenarios to consider that can delay transactions (not enough funds within the routing path) or withdraw funds (uncooperative behaviors for channel closure).	Commercial solutions claimed to have achieved a peak of 7,200 TPS. Currently, throughput on Plasma platforms is around 175 TPS: this result has been possible by using a different consensus mechanism on the Plasma chain, in addition to a shorter block time and a higher block gas limit with respect to Ethereum.	Thanks to compression, Rollups achieve theoretical maximum throughput limits of 4,500 TPS (ZK Rollup) and 800 TPS (Optimistic Rollup). However, practice shows that it is difficult for all optimality conditions to occur, reaching about 30% of the theoretical throughput.

Continue in the next page

Security	The use of hashlock and timelock mechanisms guarantees security. HTLC overcomes the problem of uncooperative behaviors. Moreover, malicious behavior is mitigated, as the attacker risks losing all funds.	Security is achieved through merkle root commitments to L1 that prevent operator's malicious behavior (unlike traditional sidechains). Most concerns are related to block withholding attack and the mass-exit problem, which are still open issues.	Different system for each solution. In particular, ZK is based on the concept of *validity proof*, using the SNARK system: it couples each batch with proof of its validity (easy to verify but difficult to generate). Optimistic Rollups, like Plasma, are based on fraud proofs: the idea is that anyone can provide proof of an incorrect batch which will then be verified as well.
Decentralization	Fully guaranteed from a theoretical perspective. However, the real structure consists of a few active channels between users and a limited number of *hubs* connected to a large number of nodes.	Consensus mechanisms exploited in Plasma are usually less decentralized than L1, in order to achieve better performance. In practice, a trade-off between decentralization and performance must be found, for example by using PoS on L2.	Although on the one hand the Rollups have a high decentralization guaranteed by the main chain (albeit in summary form, all transactions are stored in the blockchain), on the other, the transactions are managed by a limited number of smart contracts, effectively increasing centralization.

Privacy	High level of confidentiality: transactions on direct channels are completely private, while the level of confidentiality with intermediaries increases as the length of the path increases.	Not a primary concern in Plasma, but since the concept is avoiding to broadcast all the transactions to the root chain, the privacy heavily depends on the design of the Plasma child chain.	Surely, it is the solution that guarantees less privacy among the analyzed. Precisely, due to the fact that a summary of all transactions is published in the main chain, the transactions somehow "remain public" (at least in part, from which the entire transaction can be traced).
Fees and micropayments	Transactions in a direct channel are not subject to fees, while intermediate nodes that route indirect payments generally ask for them as an incentive. At the moment of writing, the additional cost for transaction (multi-hop scenario) is very low. L1 fees are present for channel opening and closure.	Transactions on L2 are much cheaper than those on L1, because a Plasma chain can exploit a less decentralized consensus mechanism. Nevertheless, even if low, they are still needed to discourage spam attacks on the Plasma chain. L1 fees are still needed to perform deposits and withdrawals.	The best case occurs with ZK Rollup: it implies negligible *off-chain* fees ($\sim$0.001 USD for transfers, higher for withdrawals). *On-chain* fees are still required for deposits. Optimistic Rollups have higher fees compared to ZK, even if they are still low if compared to L1.

8 Layer 2 Use Case: Supermarket

In order to be able to test the described Layer 2 solutions in practice, a proof of concept of a suitable use case for these technologies can be proposed. The application simulates a possible architecture put in place by a supermarket chain, to allow customers to pay using cryptocurrencies.

The factors under analysis for each Layer 2 option are the following;

- The commitment that the shop should undertake to implement the solution, both in terms of software engineering effort and costs
- Usability for customers
- The achieved performance (throughput and latency)

Basically, this sample application implies the creation of a parallel payment system with respect to pre-existing ones. This therefore falls into the case of micropayments, which have been detailed in the previous sections regarding each solution.

The first of the previously described parameters also includes a brief analysis about the current state of the art of tools for a specific L2 platform. As a matter of fact, this is crucial to determine how much development effort is required by the owner of the system, and it is an indication of how mature the platform ecosystem is.

In this context, the term *usability* is referred to a combination of two main aspects: the ease of use for customers and the cost per transaction.

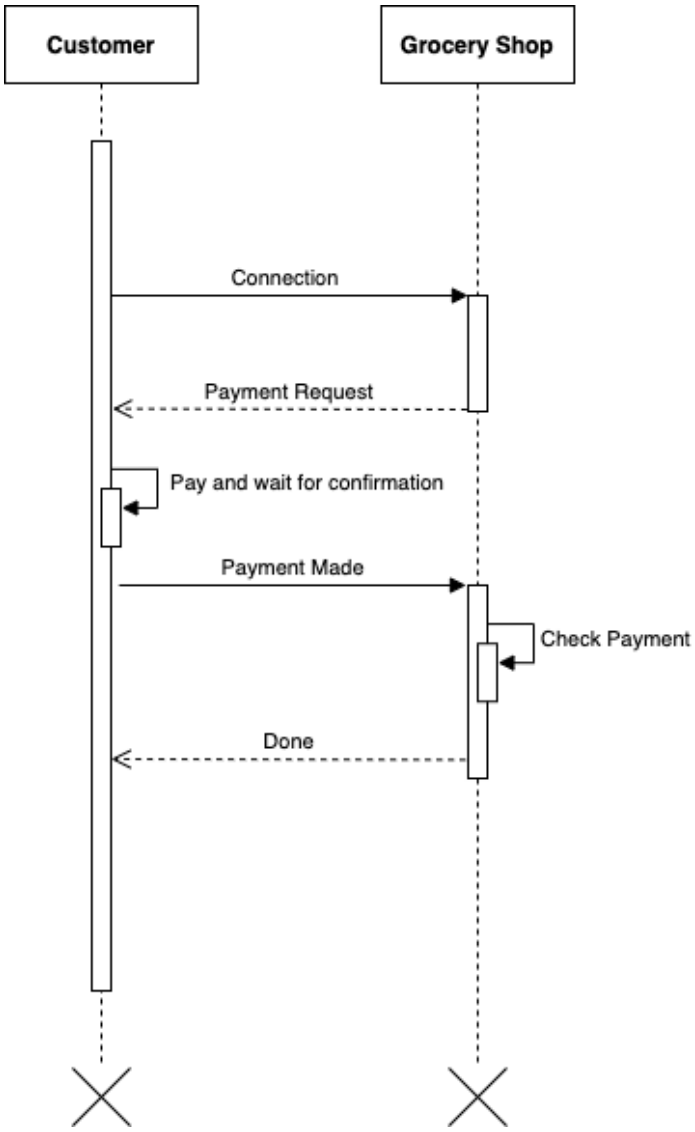
Regarding performance, in this case an analysis in terms of throughput (TPS) is not enough: what has to be taken into account is also the transaction latency, i.e. the time that each single user has to wait in order to have its payment confirmed by the underlying architecture. In fact, a payment method implying an average confirmation time of, for instance, 10 minutes, is not usable in a retail context.

It is important to point out that the results in terms of throughput that will be shown in the following sections are related to the number of transactions per second that were dedicated only to the application, not the overall performance of the tested platforms. Furthermore, the performance analysis presented in this work should be considered only as an indication of the performance of each Layer 2 solution; this is caused by the fact that *benchmarking* this type of applications on public testing networks leads to results that are not always reproducible and extremely variable (mainly due to changes in the level of congestion and average fees), especially if performed using external API providers.

Theoretical results have been discussed in previous specific sections (section 4.1 LN, section 5.1 Plasma, section 6.1 Rollups). For an in-depth analysis we refer to other works designed to measure this type of statistics more thoroughly.

Proposed Schema

The proposed flow for the proof of concept is shown below.



For all Layer 2 solutions under analysis, interaction between customers and the supermarket has been emulated by using the *socket.io* Node.js library to implement real-time communication. In a real-world scenario, the initial setup needed for the communication would be replaced by other systems, such as a QR code shown by the cashier to the customer; the latter would then use a mobile application released by the supermarket, which in practice would be a Layer 2 cryptocurrency wallet.

This schema would allow the system to remain as decentralized as possible, because private keys are kept locally on users' devices, and customers have full control of their funds.

Common usability drawbacks

Decentralization introduces some disadvantages in terms of usability that are common to all the solutions shown below:

- Users must be able to securely manage private keys and cryptocurrency wallets
- The need for customers to handle deposits and withdrawals, with the resulting fees *on-chain* to be payed by users

Estimated performance requirements

To evaluate the various frameworks exploited in the following sections, an estimated target throughput has been computed. A nationwide supermarket with the following properties was considered:

- Approximately 400 stores with 10 cash registers each
- For every cash register, a payment every 2 minutes has to be processed

Therefore, it is possible to estimate the performance needed for each store to be around 0.083 TPS and the dedicated throughput needed for an application of this type to be:

$$0.083\,TPS \cdot 400 \approx 33\,TPS$$

The tests whose results will be shown below were performed by trying to process 200 transactions for each execution.

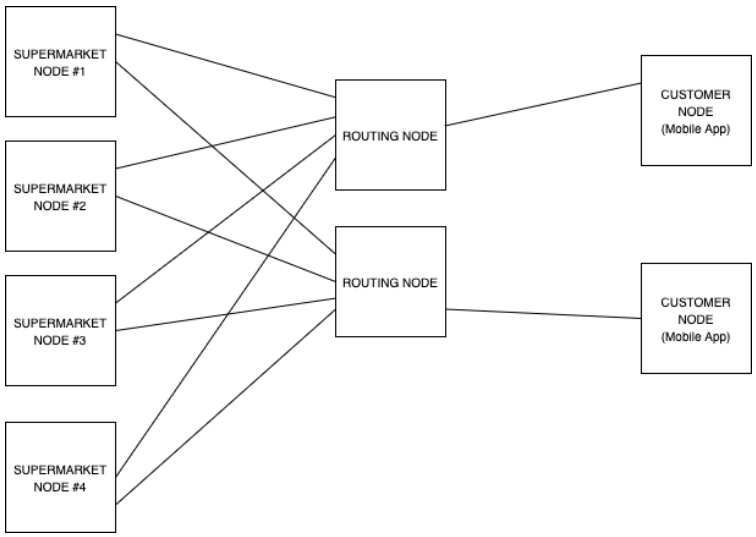
8.1 Lightning Network

8.1.1 Development effort and costs

Each Lightning Network (LN) user must have his own node, in order to manage channels and make payments. At the time of writing, current commercial mobile LN wallets (e.g., *Eclair Mobile*) include a running instance of a LN client. Then, since each LN node instance must also be connected to a Bitcoin node, usually the application is connected to a remote Bitcoin node instance. This is a trade-off needed to balance privacy and usability of the application, by avoiding the need for the user

to have enough storage space and computational resources to maintain a local Bitcoin node.

A possible architecture for the system could be designed as follows.



In this context, lines between nodes represent open channels. Routing nodes are third-party *hubs* that are willing to offer routing as a service.

This schema, not implying a direct LN channel between grocery shop nodes and each customer, would allow the supermarket to close a limited number of channels (those open with the routing nodes) in order to withdraw funds to Bitcoin L1: the reduced number of channels to be closed mitigates costs for withdrawals and avoids the risk to congest the L1 (when closing channels); as an example, the supermarket could decide to close channels and reopen them periodically (once a week, or when the received amount is above a predefined threshold).

Summarizing, the development of such a system running on top of Lightning Network would need:

- A set of Bitcoin nodes and Lightning Network nodes maintained by the supermarket, to receive payments
- A mobile wallet application to allow users to make payments
- Back-end services capable of generating invoices and verifying if a Lightning Network payment was successful, in order to conclude the transaction with the customer

Speaking about transfer fees, in this case they apply since payments by users have to pass through third party routing nodes. As in the case, today, with credit card fees,

it is reasonable to assume that these costs would be paid by the supermarket (by discounting the amount requested to the user). To quantify these costs, data from *1ML.com* can be considered: the service estimates the average fee for every routing node that a transfer has to go by to be around 1 satoshi (which is currently equivalent to ∼0.00035 USD).

8.1.2 Customers usability

Current LN wallet applications require a minimum knowledge regarding channels and their opening and closing mechanisms. This procedure could be made semi-automatic with a mobile app dedicated to supermarkets, for example with a user wizard to open a channel with a LN *hub* used by the system; nevertheless, the customer still has to be conscious of what he/she is doing, and the costs related to every operation. Regarding costs, what the customer must pay *on-chain* are the fees to open and close the channel with the routing nodes. At the time of writing, costs for users are the following:

$$Opening\ Channel\ Transaction\ size \approx 235\ bytes$$

$$Opening\ Channel\ fee \approx 235\ bytes \cdot 100\ satoshis/byte \approx 23,500\ satoshis = 0.000235\ BTC$$

$$Closing\ Channel\ Transaction\ size \approx 300\ bytes$$

$$Closing\ Channel\ fee \approx 300\ bytes \cdot 100\ satoshis/byte \approx 30,000\ satoshis = 0.0003\ BTC$$

These fees would need to be paid by customers every time they want to *top up* their wallet or when they want to withdraw funds to L1. Currently, it is not possible to add funds to an already existing channel: this means that, if the channel between the user and the routing node runs out of funds, it needs to be closed and reopened again. Some projects are trying to solve this problem (e.g. *Lightning Loop*), but they are still in a beta version.

8.1.3 Performance

The implemented architecture is the one described before, using a single routing node between customers node and supermarket node. On Lightning Network Testnet / Bitcoin Testnet (with *lnd* as Lightning node implementation) the obtained results are shown below.[11]

[11] Details about the performed tests and the proof of concept source code can be found at `https://github.com/CosimoSguanci/Blockchain-Layer-2-Proof-of-Concept-App-Polimi`

Date	Time	TPS	Latency (ms)
20-05-2021	08:00 AM	60.46	943.79
20-05-2021	09:00 AM	63.96	970.2
20-05-2021	04:00 PM	73.80	1095.40
20-05-2021	05:00 PM	51.13	888.99
20-05-2021	09:00 PM	57.45	904.96
20-05-2021	09:30 PM	66.49	1062.9

Date	Time	TPS	Latency (ms)
05-06-2021	10:00 AM	66.77	903.93
05-06-2021	12:00 PM	54.36	911.66
05-06-2021	06:00 PM	75.38	1128.67
05-06-2021	07:00 PM	61.18	873.11
05-06-2021	10:00 PM	68.99	1075.01
05-06-2021	10:30 PM	60.62	963.51

On average, the throughput and transaction latency observed are the following.

Latency	Throughput
976.84 ms	63.38 TPS

"Benchmark" tests were able to process around 63 TPS on average. This result is undoubtedly encouraging, because it shows that the technology has the potentiality to support the presented kind of application, also considering transaction latency: a confirmation waiting time that is less than 2 seconds makes cryptocurrencies a suitable payment method for most real-world scenarios.

8.2 Plasma

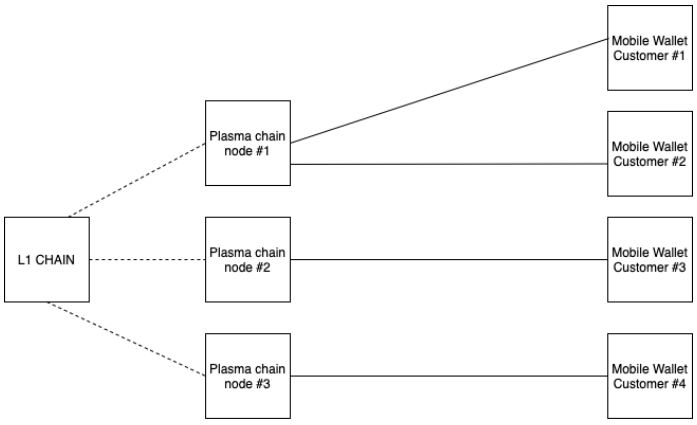
In order to implement the proof of concept application on Plasma, *Polygon* has been used; it is an adapted account-based implementation of Plasma, enriched with a PoS checkpoint layer.

8.2.1 Development effort and costs

Since *Polygon* runs on top of the EVM, in principle each service which runs on an Ethereum-like blockchain is compatible with it: it is enough to change the RPC connection to a node of the Plasma network. In practice, this kind of solution can benefit from the vast Ethereum ecosystem, in terms of libraries (e.g., *web3.js*) and resources.

Regarding blockchain nodes, although some services offer RPC APIs, the most reliable solution would be to setup a set of dedicated Plasma chain nodes to better support the application and also avoid traffic limits, which are usually a constraint of third party services.

The necessary architecture for the application would actually need less components than the one described for Lightning Network, as shown in the following picture.



Here, solid lines represent simple HTTP/WebSocket connections, while dashed lines are merkle commitment of blocks to the L1. The application would consist in an Ethereum-like mobile wallet, to allow users to send transactions on the L2 chain.

8.2.2 Customers usability

As stated in previous sections, since the goal is to achieve a decentralized solution, the platform should be *non-custodial*, i.e. only the user has his own private keys. Therefore, the user must be able to securely keep credentials to avoid the risk of losing money. Then, since this application would be specific to the supermarket use case, transfers should be easy to carry out and, therefore, accessible to all customers (this is a problem that should be solved by a proper UX/UI).

Regarding costs, payments on a Plasma chain are not free, but are usually very convenient with respect to L1 transactions, due to the different consensus mechanism. At the time of writing, the average gas price on *Polygon* is 3 Gwei: cost for a transfer (that is independent from the transacted amount) is computed as follows:

$$Native\ currency\ transfer\ gas\ needed = 21,000$$

$$Average\ Gas\ Price \approx 3\ Gwei = 3,000,000,000\ Wei$$

$$Transaction\ cost = 21,000 \cdot 3,000,000,000\ Wei = 0.000063\ in\ native\ currency$$

These fees are needed for every transaction on the Plasma chain. In the context of the supermarket proof of concept, the supermarket could take charge of these costs by discounting the amount owed by the user.

Fees on L1 are needed only in case of deposit/withdrawal between L1 and L2 and are shown below (considering 40 Gwei as "standard" gas price, taken from *ETH Gas Station* at the time of writing).

$$Current\ average\ mainnet\ Gas\ Price \approx 40\ Gwei = 40,000,000,000\ Wei$$

$$Gas\ needed\ for\ ETH\ deposit\ on\ Plasma \approx 77,000$$

$$ETH\ Deposit\ cost \approx 77,000 \cdot 40,000,000,000\ Wei = 0.00308\ ETH$$

$$Gas\ needed\ for\ ETH\ withdraw\ from\ Plasma \approx 245,000$$

$$ETH\ Withdraw\ cost \approx 245,000 \cdot 40,000,000,000\ Wei = 0.0098\ ETH$$

In addition to costs, another constraint for withdrawals is the 7 day dispute period that users must wait before being able to retrieve their funds on the root chain.

8.2.3 Performance

Tests have been carried out on the *Polygon "Mumbai"* testnet, which has Ethereum's *Görli* testnet as L1 root chain, and the native currency has been involved in transfers. Transactions have been "bundled" in a single RPC call by exploiting *web3.js BatchRequest* functionality.

Date	**Time**	**TPS**	**Latency (ms)**
20-05-2021	08:00 AM	36.49	3688.968
20-05-2021	09:00 AM	35.72	3579.16
20-05-2021	04:00 PM	34.87	3625.5
20-05-2021	05:00 PM	35.18	3694.74
20-05-2021	09:00 PM	47.82	3657.12
20-05-2021	09:30 PM	37.44	3648.35

Date	**Time**	**TPS**	**Latency (ms)**
05-06-2021	10:00 AM	35.41	3642.1
05-06-2021	12:00 PM	34.78	3625
05-06-2021	06:00 PM	39.39	3698.01
05-06-2021	07:00 PM	39.73	3625.9
05-06-2021	10:00 PM	43.20	3714.44
05-06-2021	10:30 PM	26.69	3664.43

Latency	**Throughput**
3,625.31 ms	37.22 TPS

This case refers to transaction finality on L2. In *Polygon*, finality on-chain (merkle root commitments) is achieved periodically at intervals ranging from 15 minutes to 1 hour.

The obtained performance is worse than Lightning Network. Some possible explanations for this result include the fact that the *Polygon* node client (which is based on *Geth*) may need further optimizations in order to stably support very low block times (∼2 seconds on *Polygon*). Moreover, in this context performance heavily depends on the RPC service provider (for this proof of concept, *Infura* and *BlockVigil* were used).

Anyway, these results still seem to be compatible with the supermarket use case, although better latency is desirable. Considering that the platform used for this test

is a Plasma chain which is *shared* with other applications, it is possible to envision a scenario of a dedicated Plasma chain for the supermarket use case, which would lead to better performance (at a cost of increased setup effort and maintenance costs from the supermarket perspective).

8.3 Rollups

To showcase the potentiality of Rollups technology, *zkSync*, a ZK Rollup solution proposed by *Matter Labs*, has been used.

8.3.1 Development effort and costs

At the time of writing, *zkSync* is not EVM-compatible (an EVM-based programming model is scheduled to be introduced with *zkSync 2.0*). For this reason, existing software interacting with the Ethereum ecosystem cannot be turned into a *zkSync* application by simply changing a RPC endpoint, and *zkSync* offers libraries supporting the most popular programming languages. The architectural design in this context is easier than the previous two: at the time of writing, becoming a validator of the platform does not seem possible, hence the mobile application would communicate to the *rollups* platform by making use of *zkSync* libraries and APIs. Validators will then publish the validity proof *on-chain*.

8.3.2 Customers usability

From a usability perspective, some common concepts with respect to Lightning Network and Plasma are present, such as those of deposits and withdrawals. To keep the system fully decentralized, users should be able to move their funds to the *rollup* contract *on-chain*, in order to be able to use them on L2.

Transaction costs are dramatically lower than L1. Considering Ethereum mainnet as L1 and *zkSync* as L2, ∼8x cheaper transfer transactions on L2 (performing a conservative analysis by using the minimum gas price provided by *ETH Gas Station*) can be observed. A comparison of minimum fees (at the time of writing) for transfers on *zkSync* and Ethereum mainnet follows:

$$Current\ mainnet\ ETH\ transfer\ minimum\ fee \approx 0.000567\ ETH$$

$$Current\ zkSync\ ETH\ transfer\ minimum\ fee \approx 0.0000736\ ETH$$

Minimum fee data for *zkSync* was gathered by using the JSON RPC method `get_tx_fee`, offered by *zkSync* APIs.

Moreover, through the use of *Batched Transactions*, bundles including many transactions were created, with fees paid by a single sender. This feature fits in particular

with the supermarket use case, since an ecosystem in which the supermarket pays for transfer fees (as in the case of traditional payment methods) can be envisioned. In particular, 10 transfer transactions were bundled and executed, with ∼0.0001084 ETH paid in fees by a single account.

With respect to deposits and withdrawals, the former only requires an *on-chain* fee, while the latter implies an *off-chain* fee, which at the time of writing can be quantified as ∼0.0029 ETH. This fee needs to be paid by customers when they want to exit the L2 and return to the root chain. *On-chain* fees for deposits are quantified below (considering ETH as currency and 27 Gwei as the "standard" gas price):

$$Gas\ used\ for\ ETH\ deposit \approx 62,500$$

$$Current\ ETH\ deposit\ minimum\ fee \approx 62,500 \cdot 27\ Gwei = 0.0016875\ ETH$$

ZK Rollups allow for ∼10 minutes withdrawal times, which, if compared to Plasma and Optimistic Rollups dispute period, is a huge improvement in usability from the user perspective. Another advantage in usability is given by the fact that transfer fees can be paid with the same token that is being transferred, thus eliminating the constraint of having the native currency of the chain to pay fees, as it happens on L1 and Plasma.

8.3.3 Performance

In the performed tests, the *Rinkeby* Ethereum testnet was used (transactions here are represented by ETH transfers). Similar to what was done for Plasma (*Polygon*), finality is considered on L2; with high volumes, the proof time generation is expected to be around 10 minutes, i.e. every 10 minutes a validity proof is posted *on-chain* and the transactions included in it can be considered finalized.

Date	Time	TPS	Latency (ms)
20-05-2021	08:00 AM	23.19	2822.66
20-05-2021	09:00 AM	19.18	2816.14
20-05-2021	04:00 PM	34.17	2888.18
20-05-2021	05:00 PM	25.49	2810.06
20-05-2021	09:00 PM	24.92	2682.78
20-05-2021	09:30 PM	18.11	3001.98

Date	Time	TPS	Latency (ms)
05-06-2021	10:00 AM	3.08	2859.34
05-06-2021	12:00 PM	2.26	2779.62
05-06-2021	06:00 PM	2.86	2755.37
05-06-2021	07:00 PM	3.19	2793.85
05-06-2021	10:00 PM	3.78	2887.31
05-06-2021	10:30 PM	2.98	2963.27

Latency	Throughput
2,838.38 ms	13.6 TPS

Using this kind of solution, a great level of variability in terms of TPS, with respect to the other tested L2 solutions, was observed. The reason could be due to multiple causes: for sure, one of the main differences of this solution compared to LN and *Polygon* (from the point of view of an application designer) is the fact that *zkSync* APIs must be used, as, at the time of writing, it seems to be the only way to interact with the system; this "constraint" can lead performance to be highly dependent on some factors, such as APIs rate limits, or simply changes on the back-end side, released by the solution designers. Moreover, some inherent limitations are present, such as the fact that, at the moment of writing, transaction batches allow a maximum of 10 transaction authors per batch. Therefore, these results should not be considered as the best performance obtainable using a ZK Rollup solution. An interesting fact to notice is that, unlike throughput, transaction latency seems to be stable and also low, especially if compared to the Plasma use case explained in the previous section.

To summarize, it is important to point out the fact that improvements for this solution are released very frequently, so anybody can expect it to reach the necessary "maturity" to be used reliably, in the context of this proof of concept, in the near future.

8.4 Use case conclusions

With the presented proof of concept, a possible practical comparison framework has been shown, both from the point of view of a service provider (the supermarket) and from the perspective of the service users (customers). Finally, regarding performance, it is worth to compare results achieved in previous sections with the current state of one of the most popular blockchain platforms, Ethereum. Running the same benchmark tests on the Ethereum *Ropsten* testnet (which exploits PoW consensus), the following output has been obtained.

Date	**Time**	**TPS**	**Latency (ms)**
20-05-2021	08:00 AM	12.82	7984.84
20-05-2021	09:00 AM	18.8	9081.68
20-05-2021	04:00 PM	6.28	33429.86
20-05-2021	05:00 PM	10.54	17026.34
20-05-2021	09:00 PM	7.135	17327.23
20-05-2021	09:30 PM	2.26	23587.22

Date	**Time**	**TPS**	**Latency (ms)**
05-06-2021	10:00 AM	3.29	55804.78
05-06-2021	12:00 PM	4.37	70302.67
05-06-2021	06:00 PM	5.45	41491.75
05-06-2021	07:00 PM	1.99	47401.36
05-06-2021	10:00 PM	2.12	38945.72
05-06-2021	10:30 PM	5.68	38654.89

Latency	**Throughput**
33,419 ms	6.73 TPS

In the worst-case scenario for the proof of concept, ∼2x improvement in terms of throughput, and ∼9x performance improvement for transaction latency, were obtained.

In this context, Lightning Network currently seems to be the most mature solution from a performance point of view, since it allows for a substantial throughput and latency improvement. Anyway, it must be said that, as shown, probably other tools require less effort from an architectural/software engineering point of view.

The last necessary point to highlight is that all the tools that have been used are under development, therefore their features may change and/or improve over time.

9 Conclusion

The proposed work has introduced and commented blockchain scalability problem and the most popular solutions to it; in particular, Lightning Network (for Bitcoin) and Raiden Network, Plasma, ZK Rollups and Optimistic Rollups (for Ethereum) have been analyzed, both under their theoretical aspects and five comparison measures (namely scalability, security, decentralization, privacy, fees and micropayments). In order to recap, payment channels solutions, such as LN and RN, are able to achieve highly private transactions, in addition to big improvements to throughput (at least in theory); Plasma, instead, is an extension of traditional sidechains with improvements from the point of view of security and decentralization, and therefore shows interesting properties, especially because its main concepts are not so far from blockchain traditional ones (while state channels proposal, for example, is quite far from them); finally, Rollups introduce the idea of compression (opposed, in a certain sense, to that of reporting *on-chain* just a summary of many *off-chain* transactions), which is an important element for scalability improvements. Moreover, the mentioned Layer 2 technologies could also be used simultaneously on the same platform: Lightning Network on top of Plasma is a possible architecture of this type discussed by Buterin and Poon in the original Plasma proposal.

After the theoretical and comparative analysis, the supermarket use case has been described, in order to show a possible scenario of massive adoption of Layer 2 solutions (especially related to micropayments): the obtained results show how, under latency and throughput points of view, all the frameworks perform better than a "pure" L1 application (considering Bitcoin and Ethereum).

Lightning Network constitutes the most mature technology for scaling Bitcoin and, in spite of the *hub and spoke* current architecture that can partly menace some of its theoretical features, it will surely play a crucial role for blockchain massive adoption.

For what regards Ethereum, instead, the reasoning is a bit more complicated, because the community, in addition to Layer 2 proposals, is heavily pushing on Eth2 development as main scaling solution. Ethereum 2.0 (Eth2) is "a set of upgrades that improve the scalability, security, and sustainability of Ethereum"[12] by introducing proof-of-stake (PoS) and sharding; in terms of throughput, developers' goal is that of achieving thousands of transactions per second.[13] The concern, now, is to understand whether Eth2 will cause uselessness of all Ethereum L2 solutions presented in this

[12]Quotation from `https://ethereum.org/en/eth2/`

[13]Going into Ethereum 2.0 details is not one of the goals of this work; the interested reader can find useful information at `https://ethereum.org/en/eth2/` and `https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/eth-2.0-phases/`

work; in order to face the problem, it is important to point out that Ethereum 2.0 is estimated to be ready for full launch in more than one year (at the moment of writing) and that, in any case, Layer 2 technologies can integrate on top of it, as well; so, adapted implementations of Plasma and Rollups are likely to be the most important scaling solutions for some time, then they will continue to have a key role for higher throughput (in general, but especially in the early phases of Eth2 adoption). As a matter of fact, as stated by Vitalik Buterin "the scalability gains from the Layer 1 improvements and Layer 2 improvements do ultimately multiply with each other". Therefore, it is possible to consider sharding and existing L2 solutions as complementary, as performance improvements add up to each other.

Raiden Network, instead, is still an immature technology (at the moment of writing) and, according to most recent estimations, will probably have few time to settle up before Ethereum updates; in any case, it would offer unique privacy features that users might look for when transacting: this fact is probably going to lead to RN adoption inside Ethereum ecosystem, independently of throughput and Eth2 achievements.

In conclusion, Layer 2 solutions certainly represent the present and future for solving blockchain scalability problem, without renouncing to a secure and decentralized architecture.

References

[1] S. Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System". In: *bitcoin.org* (2008). URL: https://bitcoin.org/bitcoin.pdf.

[2] "The Blockchain Generations". In: *Ledger Academy* (2021). URL: https://www.ledger.com/academy/blockchain/web-3-the-three-blockchain-generations.

[3] L. Kenny. "The Blockchain Scalability Problem & the Race for Visa-Like Transaction Speed. Yes, blockchain has a scalability problem. Here's what it is, and here's what people are doing to solve it". In: *Medium* (2019). URL: https://towardsdatascience.com/the-blockchain-scalability-problem-the-race-for-visa-like-transaction-speed-5cce48f9d44.

[4] G. Del Monte, D. Pennino, and M. Pizzonia. "Scaling Blockchains Without Giving up Decentralization and Security. A Solution to the Blockchain Scalability Trilemma". In: *arXiv* (2020). URL: https://arxiv.org/pdf/2005.06665.pdf.

[5] "Blockchain Scalability - Sidechains and Payment Channels". In: *Binance Academy* (2021). URL: https://academy.binance.com/en/articles/blockchain-scalability-sidechains-and-payment-channels.

[6] A. Paszke. *Layer 2*. URL: https://academy.binance.com/en/glossary/layer-2.

[7] *Layer 2 Rollups*. URL: https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/.

[8] *Ethereum (ETH) Blockchain Explorer*. URL: https://etherscan.io/.

[9] L. Mearian. "Sharding: What it is and why many blockchain protocols rely on it". In: *Computerworld* (2019). URL: https://www.computerworld.com/article/3336187/sharding-what-it-is-and-why-so-many-blockchain-protocols-rely-on-it.html.

[10] S. Martinazzi and A. Flori. "The evolving topology of the Lightning Network: Centralization, efficiency, robustness, synchronization, and anonymity". In: *PLoS ONE* (2020).

[11] J. H. Lin et al. "Lightning network: a second path towards centralisation of the Bitcoin economy". In: *New Journal of Physics* 22 (2020).

[12] "A Beginner's Guide to Bitcoin's Lightning Network". In: *Binance Academy* (2021). URL: https://academy.binance.com/en/articles/what-is-lightning-network.

[13] J. Herrera-Joancomartí et al. "On the Difficulty of Hiding the Balance of Lightning Network Channels". In: *IACR* (2019). URL: https://eprint.iacr.org/2019/328.pdf.

[14] F. Béres, I. A. Seres, and A. A. Benczúr. "A Cryptoeconomic Traffic Analysis of Bitcoin's Lightning Network". In: *arXiv* (2020).

[15] J. Poon and T. Dryja. "The Bitcoin Lightning Network. Scalable Off-Chain Instant Payments". In: (2016). URL: https://lightning.network/lightning-network-paper.pdf.

[16] *What is the Raiden Network?* URL: https://raiden.network/101.html.

[17] J. Poon and V. Buterin. "Plasma. Scalable Autonomous Smart Contracts". In: *plasma.io* (2017). URL: https://plasma.io/plasma-deprecated.pdf.

[18] "Plasma". In: *Ethhub* (2021). URL: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/plasma/.

[19] J. F. Caracciolo. "Plasma For Dummies". In: *Medium* (2018). URL: https://medium.com/@juanfrancoc/plasma-for-dummies-a62578cddf8e.

[20] "Privacy and Scalability Solution for Ethereum Blockchain". In: *Clever Solution* (2019). URL: https://clever-solution.com/scalability-opporty-plasma-cash/.

[21] "7,200 TPS Achieved on Matic Network's Counter Stake Testnet!" In: *Matic Blog* (2020). URL: https://blog.matic.network/7200-tps-achieved-on-matic-networks-counter-stake-testnet/.

[22] T. Schaffner. *Scaling Public Blockchains. A comprehensive analysis of optimistic and zero-knowledge rollups*. University of Basel, 2021. URL: https://wwz.unibas.ch/fileadmin/user_upload/wwz/00_Professuren/Schaer_DLTFintech/Lehre/Tobias_Schaffner_Masterthesis.pdf.

[23] V. Buterin. "An Incomplete Guide to Rollups". In: *Vitalik Buterin website* (2021). URL: https://vitalik.ca/general/2021/01/05/rollup.html.

[24] "ZK-Rollups". In: *Ethhub* (2021). URL: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-rollups/.

[25] "Optimistic Rollups". In: *Ethhub* (2021). URL: https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/optimistic_rollups/.

[26] A. Gluchowski. "Optimistic vs. ZK Rollup: Deep Dive". In: *Medium* (2019). URL: https://medium.com/matter-labs/optimistic-vs-zk-rollup-deep-dive-ea141e71e075.

[27] *Lightning Network Statistics*. URL: https://1ml.com/statistics.

[28] *Bitcoin Average Transaction Fee*. URL: https://ycharts.com/indicators/bitcoin_average_transaction_fee#.

[29] *Mainnet Matic Explorer*. URL: https://explorer-mainnet.maticvigil.com/.

[30] "Till it's lightning-fast. Uncover the Lightning Network Transactions". In: *Medium* (2019). URL: https://medium.com/@yyforyongyu/till-its-lightning-fast-uncover-the-lightning-network-transactions-f3180e467857.

[31] E. Felten. "What's up with Rollup". In: *Medium* (2019). URL: https://medium.com/offchainlabs/whats-up-with-rollup-db8cd93b314e.

[32] *zkSync FAQ*. URL: https://zksync.io/faq/.

[33] *zkSync Mainnet Explorer*. URL: https://zkscan.io/.

[34] *Lightning Network Daemon*. URL: https://github.com/lightningnetwork/lnd.

[35] *Lightning Network Service*. URL: https://github.com/alexbosworth/ln-service.

[36] *Matic Network Documentation*. URL: https://docs.matic.network/.

[37] *Matic Network Basic FAQ*. URL: https://docs.matic.network/docs/faq/faq/.

[38] *Introduction to zkSync for Developers*. URL: https://zksync.io/dev/#overview.

[39] "Understanding Ethereum Scaling: Categorizing projects by approach adopted". In: *Matic Blog* (2019). URL: https://blog.matic.network/understanding-ethereum-scaling%E2%80%8A-%E2%80%8Acategorizing-projects-by-approach-adopted/.

[40] "What Is Ethereum 2.0 And Why Does It Matter?" In: *Binance Academy* (2021). URL: https://academy.binance.com/en/articles/what-is-ethereum-2-0-and-why-does-it-matter.

[41] *The Eth2 upgrades*. URL: https://ethereum.org/en/eth2/.

future internet

MDPI

Review

Overview of Blockchain Oracle Research

Giulio Caldarelli

Department of Business Administration, University of Verona, 37129 Verona, Italy; giulio.caldarelli@univr.it

Abstract: Whereas the use of distributed ledger technologies has previously been limited to cryptocurrencies, other sectors—such as healthcare, supply chain, and finance—can now benefit from them because of bitcoin scripts and smart contracts. However, these applications rely on oracles to fetch data from the real world, which cannot reproduce the trustless environment provided by blockchain networks. Despite their crucial role, academic research on blockchain oracles is still in its infancy, with few contributions and a heterogeneous approach. This study undertakes a bibliometric analysis by highlighting institutions and authors that are actively contributing to the oracle literature. Investigating blockchain oracle research state of the art, research themes, research directions, and converging studies will also be highlighted to discuss, on the one hand, current advancements in the field and, on the other hand, areas that require more investigation. The results also show that although worldwide collaboration is still lacking, various authors and institutions have been working in similar directions.

Keywords: blockchain; smart contracts; oracles; bibliometric analysis

check for updates

Citation: Caldarelli, G. Overview of Blockchain Oracle Research. *Future Internet* **2022**, *14*, 175. https://doi.org/10.3390/fi14060175

Academic Editors: Hossein Hassani and Nadejda Komendantova

Received: 5 May 2022
Accepted: 5 June 2022
Published: 8 June 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

1. Introduction

"Although oracles play a critical role . . . the underlying mechanics of oracles are vague and unexplored" [1]. A preliminary study on Decentralized Finance (DeFi) oracles from the University of Singapore shows that despite the massive amount of money managed by oracles on DeFi platforms, their functions and roles are still widely neglected. Despite the plethora of papers involving blockchains, less than 15% consider oracles, and an even smaller percentage further investigated related issues [2]. The subject of blockchain oracles is critical because the entire concept of blockchain applications revolves around the idea of decentralization and trustless transactions. Those pillars, however, are undermined when, gathering real-world data, blockchain applications rely on centralized and trusted third parties. This issue, either addressed as an oracle problem [3] or an oracle paradox [4], makes the community of blockchain enthusiasts quite skeptical about real-world applications [5]. Proposing a robust blockchain application against the oracle problem requires the redaction and discussion of the so-called "trust model", a document or scheme that broadly explains how data are fetched by oracles in a decentralized and trustless manner [6–9]. A robust trust model should first include information concerning how data collected by oracles are validated before being pushed into the smart contract. Second, it should specify how the security and unforgeability of data are ensured from the time they are collected to the moment they are permanently stored on the ledger. Third, it should outline the incentive mechanism implemented to prevent collusion or the deliberate tampering of data feeds for selfish purposes [9–11]. Defining and adopting a robust trust model is not only essential for a blockchain application to work properly but is also often considered the key to mass adoption [12]. However, academic contributions concerning oracles or those discussing a detailed "trust model" [2] remain scarce. On the one hand, proposing a real-world blockchain application without analyzing the oracle's role in depth poses serious doubts about the feasibility and genuineness of the underlying project [13]. On the other hand, proposals with a detailed trust model would greatly help researchers and

practitioners analyze oracle-related features and issues and reproduce successful projects respectively [14].

Therefore, knowing which institutions are actively undertaking research on blockchain oracles and which ones are already implementing them in real-world applications is interesting and important. Scholarly interest in blockchains has resulted in some literature reviews on this topic, but none has yet undertaken research through a bibliometric analysis on blockchain oracles [15–17]. A bibliometric analysis aims to identify how the body of knowledge on blockchain oracles has evolved in the last few years in terms of the leading publication outlets, the geographical distribution of research communities, the density of collaboration, and methodological approaches. Unlike classic literature reviews, a bibliometric analysis provides a quantitative and structural overview of the investigated scientific field, reducing the chances of subjective biases [18]. The advantages of undertaking this type of study are the representation of a phenomenon in a formal and objective way, ensuring the robustness and reproducibility of results. A bibliometric analysis is also meant to guide scholars who are interested in undertaking research in that sector to understand the research gaps, methodologies used, and appropriate outlets for publication. To ensure the significance, usability, robustness, and replicability of the research study, this paper will follow a standard bibliometric approach that has been used in several studies across different disciplines [19–23]. The methodology will be extensively explained so that any individual can reproduce every passage, regardless of their expertise. The data extracted will be motivated by the associated meaning and will be presented with the aid of figures and tables. Following prior bibliometric analyses in other sectors, the collected sample will be organized based on the categories and sub-categories of the topics [22,24]. In this study, three areas will be investigated. First, an overview of the most productive institutions (in terms of papers published), the most cited authors, and the most common publication outlets will be provided. The authors will then have a better overview of the venues that support research in this domain. Second, ongoing studies will be further investigated to identify common streams of research, themes, and research directions to incentivize cooperation and progress in the field. Third, by discussing the reviewed literature, we will highlight areas that require further investigation. The following are the objectives of the study:

Objective (1) Identify the most cited authors and productive institutions to find institutions and authors focused on the subject of the study;

Objective (2) Identify research themes, directions, and converging studies to promote cooperation and progress;

Objective (3) Highlight the areas that require further investigation.

We consider this study necessary given the massive resonance of blockchain-related research and the slight growth in oracle-related investigations [14,15]. The contributions provided in this study will help researchers and entrepreneurs know which institutions are actively involved in a specific real-world blockchain application, how oracles are implemented, and which aspects the academic studies are focusing on. Discussing the key findings of the reviewed papers can also help other academics improve the quality and speed of research in related fields [2,25]. In contrast to other bibliometric analyses in the field of blockchains, this study focuses on oracles, a specific aspect of the technology that particularly affects real-world applications. Specific bibliometric analyses on cryptocurrencies and blockchains in healthcare or supply chains already exist, but to the best of the author's knowledge, there are no studies focused on oracles yet.

To better understand the value and contribution of this paper, we should point out that real-world blockchains to which this study refers are applications other than cryptocurrencies, such as healthcare, supply chain, DeFi, and resource management. Therefore, specific studies on blockchain characteristics, ecosystems, and cryptocurrencies are not considered in this paper because they are not directly related to blockchain–oracle ecosystems. Furthermore, a certain degree of subjectivity, especially in the selected categories, cannot be excluded despite the rigorous research design. Given the absence of prior studies, a

predetermined framework was also not available to build upon. Given the scarcity of data and the increasing academic interest in the subject, the data presented in this study may also face early obsolescence.

This paper is organized as follows: Section 2 covers the literature background, and Section 3 outlines the methodology used. Section 4 summarizes the results, and Section 5 reviews the literature, identifying common themes, research directions, and converging studies. Section 6 discusses the review results and identifies areas that need further investigation. Section 7 concludes the paper by providing suggestions for further research.

2. Literature Background

The power of Bitcoin lies not only in its decentralized features but also in its programmability. Experts, such as Antonopoulos, address it as "programmable money" [26]. Just by using "scripts" and without the intervention of third parties, premade "agreements", such as timelocks, Pay-to-Script-Hash, and multi-signatures, can be executed on transactions [27]. However, because of Vitalik Buterin and the introduction of the Ethereum virtual machine with smart contracts, blockchains became more developer-friendly and could be easily programmed for applications above the simple exchange of cryptocurrencies [5]. Nonetheless, the Ethereum blockchain needs to be a closed ecosystem operating on data that are already on the blockchain to reproduce Bitcoin's trustless and deterministic setting [5]. This condition is necessary to ensure that all the required data for smart contracts are publicly verifiable and auditable by all nodes [5,28]. Without data coming from the external world, the range of possible automated contracts would have been extremely limited [29]. Therefore, a means to deliver extrinsic data to the blockchain was needed to broaden the use of smart contracts [3,30,31]. This method is called an oracle. The oracle is an entire ecosystem that permits the collection from and the transfer and insertion of external data to the decentralized application [32,33]. As displayed in Figure 1, the oracle ecosystem usually comprises the following three parts.

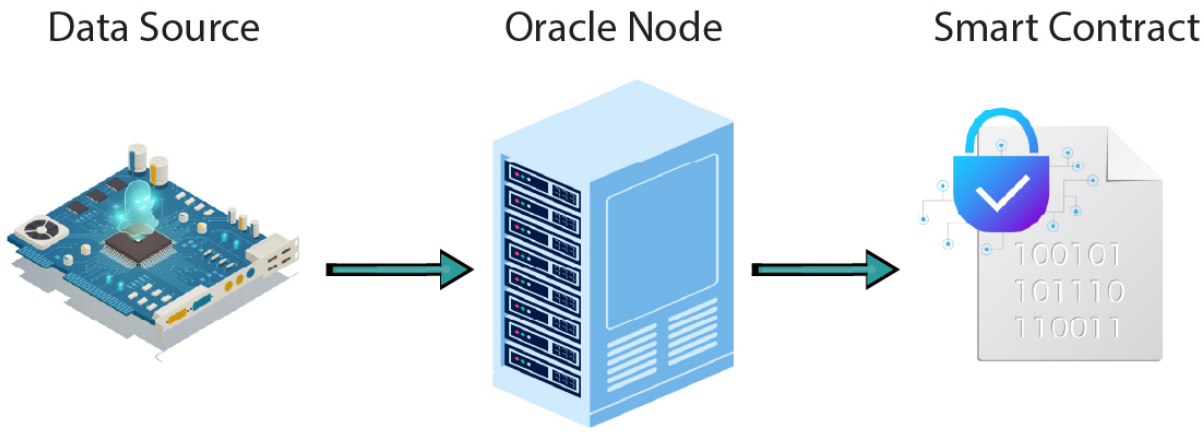


Figure 1. Oracle ecosystem.

Data Source: This is the source from which the data are collected and stored. It may or may not eventually be used by a decentralized application. The data source can be a Web Application Programming Interface (API), a sensor, or a human aware of a specific knowledge or event [34].

Communication Channel: This is usually referred to as "node". It collects data from the data source and delivers them to a smart contract so that the latter can be executed. Sometimes, oracle nodes coincide with blockchain nodes, but this is not always the case [29,35].

Smart Contract: This contains the code that establishes how the collected data can be managed. Usually, it has prespecified quality criteria for data to be accepted or rejected. If necessary, it may also perform computations to deliver the appropriate data to the contract [36,37].

Depending on how these three parts are organized and interact with each other, multiple types of oracles can be designed [12]. These three parts of an oracle are not always separate from each other, as the same entity may sometimes cover two or three roles at once. A human, for example, can serve as a data source and communicate the data directly to a smart contract [38]. In actuality, having more than one entity that covers the role of data source/node is possible and desirable. Relying on multiple entities is, in fact, crucial to ensure the execution of smart contracts, especially when one or more data sources/nodes are malfunctioning or offline [39].

The above-described oracle ecosystem is typical of blockchains that support smart contracts (e.g., Ethereum and Tron). Instead, oracles are implemented differently for blockchains, such as Bitcoin, where smart contracts (apart from a few scripts) are unavailable. If smart contracts are unavailable, oracles are usually implemented through M-of-N (e.g., three out of five) multi-signature wallets, requiring more than one signature to broadcast a transaction [40]. Therefore, the owner of a key plays the role of an oracle and executes the transaction when a certain condition is met. In that case, the oracle covers both the role of the node and the data source—for example, an agreement that sets a payment upon the delivery of a parcel (Figure 2).

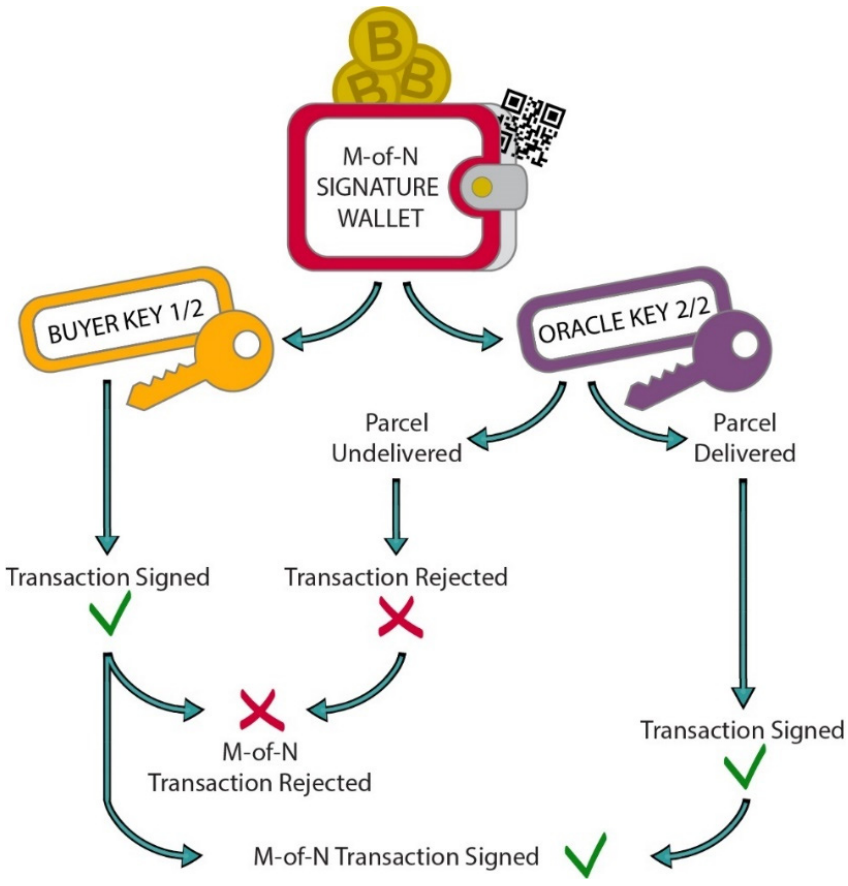


Figure 2. M-of-N Oracle Example.

A multi-signature wallet must be set up in which one of the keys has to be entrusted to a third party that performs the role of an oracle. When the buyer acquires the product, she signs the transaction with her key. However, given that the second signature has not been inserted, the transaction remains on hold. When the parcel is delivered, the entity in control of the oracle key signs the transaction, allowing for the successful execution of the transaction. Evidently, the choice of the entity that possesses the oracle key plays a crucial role in those types of ecosystems [3]. This is a trivial example of an oracle solution on the

Bitcoin blockchain implemented in traceability; however, the most commonly used cases belong to the finance/gambling field [41].

A thorough explanation of all oracle types is beyond the scope of this study; however, further information can be found in dedicated papers and web articles listed in references [10,31,40,41]. Given that oracle ecosystems operate in a different way with respect to blockchains, characteristics such as immutability, transparency, and trustless execution are not ensured [42]. This discrepancy in attributes implies that when blockchain-based applications need data from the external world, the characteristics of oracles are to be taken into serious consideration. If the data source is unreliable, the node is not trusted (or private), and the smart contract is poorly audited, the fact that an application runs on the blockchain is practically irrelevant [3,14,43]. Depending on third parties, blockchain technology alone cannot represent a solution to centralization, trust, and security issues.

This condition, widely explained by blockchain experts such as Andreas Antonopoulos and Paul Sztorc [44,45] and labeled by Dalovindj [41] as "the oracle problem", must be considered at the time of integrating blockchain with applications in the area of the supply chain, healthcare, and academic credentials. Various consequences may be faced, depending on the faulty oracle part and the application type [14,38,46]. In the healthcare sector, the presence of oracles constitutes another possible source of data breach, exposing patient records to theft or manipulation [47]. In the DeFi sector, the dependency on oracles would expose decentralized applications that rely on centralized or insecure data sources to risk millions of dollars of invested capital [46,48].

In the traceability sector, blockchain technology has been proposed, relying principally on the misconception that considering that the origin and movement of a cryptocurrency on the blockchain can be traced in a secure and trustless manner, the same can be performed with a tangible asset, such as food, clothes, and medicine [44]. Because the dependency on oracles for real-world applications makes it unlikely to reproduce the same level of tracking accuracy, only a few traceability projects show some robustness against that issue [8,42]. Lately, with Non-Fungible-Tokens (NFTs) and stablecoin technology, the blockchain-based traceability of tangible products is also following another path [49–51]. Rather than directly tracking a real product with blockchains, companies are instead creating a representation of those on the blockchain (NFTs) to guarantee genuineness and ownership.

Because of the oracle problem, numerous critiques and concerns also arise for other blockchain applications, such as intellectual property rights management, e-government, and resource management [30,52–54].

For these applications to run genuinely decentralized and trustless, oracle ecosystems should be structured to ensure the same characteristics as blockchains. However, unlike blockchain technology, which has a history and development of nearly thirty years (considering the work of Haber and Stornetta [55] as its precursor), oracle ecosystems are relatively newer and unexplored spaces with few actors and limited literature [2]. This is the gap in which this study finds its legitimacy. It aims to shed light on academic contributions concerning blockchain oracles and promote cooperation and progress.

3. Methodology

An appropriate methodology should be chosen to fulfill the purpose of this study. Furthermore, an in-depth description of the steps followed had to be provided to ensure the reproducibility of the results. A bibliometric analysis was perceived as the appropriate method for reaching the goals of this research. Moreover, its standardized and systematic approach would ensure the reproducibility of results [19,56]. Building on prior bibliometric analysis [57,58], the methodology description will first involve database selection, inclusion, and exclusion criteria and, finally, data extraction variables. Regarding data collection, the intention is to include as many articles as possible, as long as they are academic in nature. Therefore, gray literature, such as whitepapers, opinion posts, and news, will not be considered in this research. On the one hand, although not peer-reviewed, this analysis will also consider preprints. The reason for this choice is that the included preprints are written

by academics for submission to academic journals. On the other hand, non-peer-reviewed materials, such as opinion posts, are not meant to follow an academic path. Following Buttice and Ughetto [24] and Martinez-Climent et al. [56], the selected databases were Scopus and Web of Science (WoS), but Google Scholar was also queried. As the analysis also comprises preprints and unpublished manuscripts, limiting the research to Scopus and WoS would not have been a coherent choice. Including a third database would also increase the chance of retrieving other relevant articles. For the three databases, the research was conducted on 2 March 2022. When "blockchain" and "oracle" were used as keywords in the TITLE-ABS-KEY of Scopus database, 312 articles were identified. In the WoS database, two strings were implemented in the "Topic" section so that articles containing the word "oracles" were also included and identified. The research returned 143 results. Google Scholar database was queried using the same keywords as those used on the Scopus database, but the queries returned more than 10,000 entries because of their structural differences with Scopus and WoS. For that reason, and due to saturation of results, the author decided to stop the research study on Page 35 (which presents 350 entries organizing results in ten per page). Table 1 summarizes the queried databases, along with the selected research strings. Appropriate exclusion criteria were adopted to narrow down the most appropriate data sample, with the aim of balancing inclusiveness with relevance. However, no restrictions based on language or timeframe were applied because of the nascency of the topic and the research goal. Given that the goal was to gather all relevant information about oracle research, related authors, and institutions, adding a time or language restriction was not a coherent choice.

Table 1. Databases and Research Strings.

Database	Research String
Scopus	(TITLE-ABS-KEY (blockchain) AND TITLE-ABS-KEY (oracle))
Web of Science	blockchain oracle (Topic) and blockchain oracles (Topic)
Google Scholar	blockchain, oracle (anywhere in the article)

First, the abstract and introduction were read to retrieve and exclude evidently off-topic papers. Many documents were included in the sample for mentioning "random oracles" or "test oracles", which, despite a similar name, were not the oracles on which this study investigates. Other papers that mention Oracle, the name of a company, were also included, which, although involved in some blockchain projects, is again unrelated to the oracles discussed in this study. After following these steps, 163, 69, and 189 articles were removed from the Scopus, WoS, and Google Scholar samples, respectively. Given that gray literature was also retrieved from the Google Scholar sample, 7 other articles were removed because they were neither written by academics nor published in academic venues. After the duplicates were removed, the three samples were merged, obtaining a nonredundant sample of 282 entries.

With the steps mentioned above, the obtained sample was composed of papers that included the "oracle" keyword and specifically referred to the communication channels between the blockchain and the real world. However, the aim of this paper was to present the portion of literature that not only mentioned the oracles or explained their use but also offered a direct contribution to the oracle literature. Therefore, to further skim the results, all PDF articles were downloaded and inspected one by one with a word processor. All occurrences of the word "oracle" were contextualized and analyzed. The criterion was that if oracles were mentioned in the introduction or literature review but did not constitute a central part of the analysis, the article was not included in the sample. To better explain this research step, the table in Appendix A provides a list of the research and inclusion criteria.

With this criterion, nearly half of the sample (120 papers) were discarded. Therefore, the final selection was reduced to 162 entries. In summary, because of these research steps, articles that not only mentioned blockchain oracles but also discussed their role and contributed to their development were retrieved. Table 2 broadly summarizes the methodology followed.

Table 2. Research Steps.

Steps	Databases			Total
	Scopus	Web of Science	Google Scholar	
Papers are retrieved using research strings	312	143	350	805
Off-topic papers are removed	163	69	189	−421
Duplicates are removed				−102
Unrelated papers are removed				−120
Final sample				162

3.1. Data Extraction

Appropriate extraction variables (displayed in Table 3) were identified to extract as much information as possible from the selected sample. As this is probably the first bibliometric analysis on blockchain oracles, building upon existing or prior research was impossible. However, given that the aim of bibliometric analyses is relatively homogeneous, extraction variables could be taken from similar papers investigating other literature domains [24,56,59]. First, the "year of publication" was considered to place the literature within a specific timeframe, whereas the "element type" shows the most usual outlet for retrieved publications. "Authors", "institutions", and "countries" of provenance geographically contextualize the paper sample, highlighting the contributors to the academic advancements in the sector.

Table 3. Extraction variables.

Variable	Description
Category	The research field of analysis
Item Type	Journal, conference, book chapter, or preprint
Year	Year of publication
First Author	Name of the first author
Authors	Full author list
Title	Title of the paper
Citations	Google Scholar citations
Outlet	Name of the journal/conference/book
Publisher	Name of the publisher
Keywords	Indexing keywords
Country	Country of the first author
Continent	The continent of the first author
Institution	Institution of the first author
Study type	Theoretical, empirical, or review

Citations and keywords were used to analyze metrics. Finally, as in Butticè and Ughetto [24], articles were further divided based on their specific fields of analysis. This categorization of papers serves to investigate whether streams of literature exist where researchers are more contributing and others that require more attention. Although it may constitute a bias, in line with prior research, articles were associated with only one field category to avoid double entries [24]. First, two main categories were identified,

mainly to distinguish between studies concerning oracles themselves and oracles applied to other sectors.

Second, the papers were divided to further differentiate them based on their specific fields of analysis. Although inspired by related research, category selection embodies a certain degree of subjectivity. Therefore, a description of these categories, starting with the main ones, is provided hereafter.

Oracle Theory (OT): Under this category, papers specifically focused on blockchain oracles, either from a theoretical or a practical point of view, were included.

Oracle Applied (OA): This category included papers that focused on real-world applications, such as healthcare, finance, and business process management, and also provided a detailed analysis of the role of oracles in these fields with theoretical or experimental approaches.

The main categories were further divided into sub-categories. Hereafter, those that belong to OT are listed as follows.

Architecture: With an empirical or theoretical approach, papers in this category performed analyses on the oracle framework to improve technical aspects, highlight current challenges, and identify new avenues for research. Unlike proposals or OA papers, this group includes works that have investigated existing oracle schemes that are not directly applied to a specific sector.

Proposal: These papers propose new oracle frameworks that may be implemented in real-world applications. These may still be at a conceptual or prototype stage.

Oracle Problem: These articles focused on aspects related to the trustworthiness of oracles and their limits to decentralization. Whereas all papers should outline trustworthy oracle environments, the papers in this category focused on the involved actors' incentives to cheat and the consequences of a deviation on the underlying applications.

Sub-categories belonging to OA, such as healthcare and energy, are intuitive, but those that require clarification are described hereafter.

Data Management: Articles concerning the transfer of data from the real world to blockchain pertain to the main category of OT. In this field, articles that analyzed access data management for reputation, privacy, or GDPR purposes were considered. Cloud-computing-related research was filed under its own category, given that it mainly concerned data elaboration.

Finance: In this category, articles that involved oracles applied in financial applications and those that explored timeliness and gas usage of transactions were grouped. Those concerning asset management on blockchains were also included.

IoT: This category comprised papers investigating oracles as efficient IoT systems but did not refer to a specific real-world application. A paper concerning IoT in the supply chain, for example, would instead be inserted into the "supply chain and traceability" category.

Business Process Management: This category included works that proposed blockchain integration in business processes, clearly identifying the role of oracles. Although the supply chain is part of the business processes, articles specifically investigating this field were filed under their own categories.

Artificial Intelligence: Papers filed under this group concerned research toward the integration of blockchain technology into existing AI tech through the use of oracles or AI to improve oracle efficiency and reliability.

Transport: This category included papers investigating blockchain integration into intelligent vehicle development and the transport industry in general. Research on IoT device/sensors specifically implemented in the transport field were also filed in this category.

Supply Chain and Traceability: Papers investigating the benefit of integrating blockchains in the local or global supply chain belong to this category. Moreover, studies that concerned the traceability of physical products or documents were also included. Works investigating the traceability of financial assets (e.g., stocks or crypto) were included instead in the finance field.

Only the first author was taken into consideration to extract the country and institution provenance of the paper. Considering all the authors would have created a bias toward articles with a higher number of authors. We were aware that this choice may eventually affect the final results, but any other option would have yielded the same results. Regarding the authors' affiliation, the choice was to take the one declared in the last published paper to avoid the problem of double affiliation. With this criterion, some affiliations may have changed by the time the paper was published. Finally, citations were taken from Google Scholar because it was the only database in which all the papers in the sample could be retrieved. We were aware that prior studies cited in this paper utilized ad hoc programs, such as VOSviewer, for the elaboration of the result graphs. However, considering the extremely limited size of the retrieved sample, Excel tables and charts were considered to be much more intuitive. Furthermore, considering preprints from Google Scholar, software such as Bibliometrix could not be implemented. Therefore, a non-automated analysis was perceived as the most reasonable option.

4. Results

In this section of the paper, the results of the bibliometric analysis are reported. With a quantitative approach, the status and trends of the literature on blockchain oracles are shown. The analysis first covers the time and space of the research and then focuses on the outlets, authors, and field of analysis.

4.1. Number of Publications Per Year

The first academic papers considering blockchain oracles appeared in 2016 and were equally distributed among the categories "oracle theory (OT)" and "oracle applied (OA)" [60,61]. As Figure 3 shows, interest in the topic remained low until 2018. Until 2019, the number of papers concerning OT were slightly more than those discussing OA. The increase and the shift in the trend can be observable from 2019, with 2020 having four times more publications than in 2018 and 2021 having more than double the number of publications of 2019. Moreover, the number of papers regarding OA started to exceed that of OT by 2021. Although the 2022 sample concerns only the first two months, the imbalance in the number of publications appears to be confirmed. These data reveal that the topic has gained more impact and attention among academics, probably because of the higher developments of blockchain-related platforms.

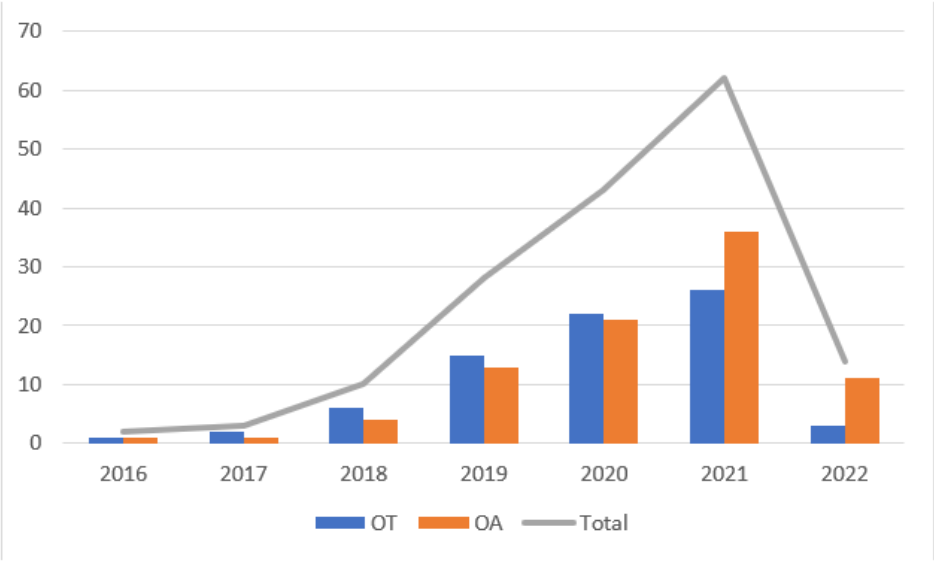


Figure 3. Publications per year.

However, in absolute terms, the overall numbers remain low, with a peak of 62 publications in 2021 and only 162 publications in all six years of academic production. These numbers show that this is still a niche subject.

4.2. Productivity Rate by Geographical Distribution

Tables 4 and 5 present the distribution of papers by country and continent, respectively. We can observe that the continents with the highest productivity are Europe and Asia, with more than 70% of total paper production. Asia, however, appears to be more focused on OA than Europe, which, although with practically the same OA contributions, presents a balance between the two main categories.

Table 4. Distribution among the ten most productive countries.

Country	OT	OA	Total
China	10	13	23
Italy	7	11	18
USA	11	4	15
Canada	7	8	15
Germany	7	7	14
UAE	1	12	13
Australia	7	2	9
France	2	3	5
Austria	4	0	4
India	2	1	3

OT = oracle theory; OA = oracle applied.

Table 5. Distribution by continent.

Continent	OT	OA	Total
Europe	31	36	67
Asia	18	35	53
America	18	12	30
Oceania	8	2	10
Africa	0	2	2

OT = oracle theory; OA = oracle applied.

Concerning countries, the situation partially reflects what is observed with continents. The most productive countries are China and Italy, followed by the USA and Canada. Only those four countries together accounted for more than 44% of total publications. Concerning fields, countries appear to be sufficiently balanced, except for the UAE, which is more focused on OA, whereas Australia, USA, and Austria mostly contribute to OT research.

4.3. Publications by Outlets and Publishers

As Figure 4 shows, the majority of papers published in this field are journals (73) and conference papers (60). However, a small portion consists of book sections (20) and preprints (9). These data contrast previous blockchain technology reviews, showing that the number of conference contributions is four times more than that of journal publications [2,16].

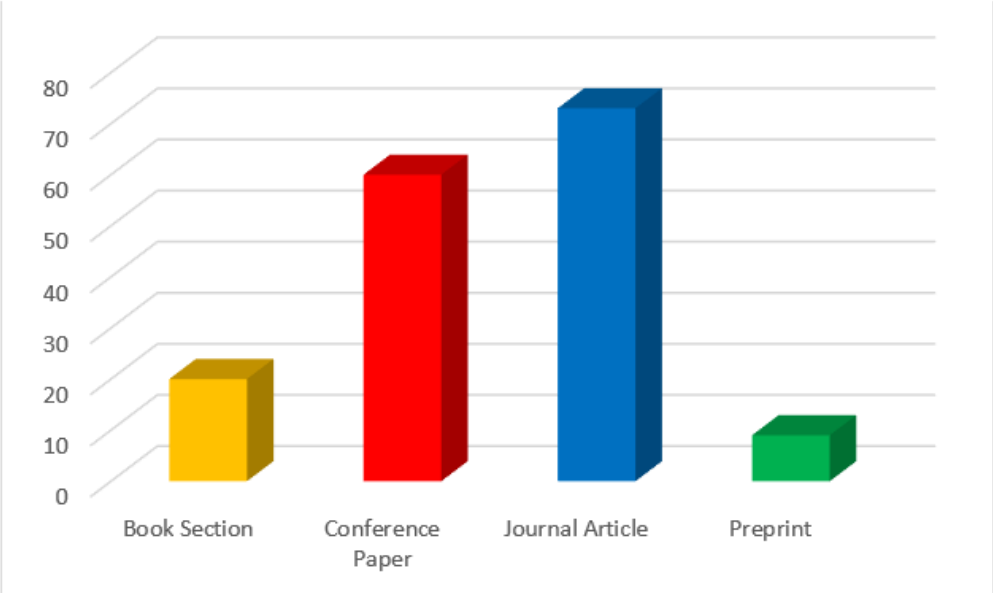


Figure 4. Publications per type.

This finding supports the idea that there seems to be no dedicated conference venue on blockchain oracles. Table 6 and Figure 5 show the distribution of papers by journal and publisher, respectively. We observed that the majority of papers (61) are published in IEEE outlets and venues, whereas 25, 15, and 13 papers are published in Springer, Elsevier, and MDPI, respectively. However, if we consider only journal publications, the weight of the contributions would slightly change, given that 43 IEEE documents were conference papers, and of 25 Springer entries, 20 were book sections.

Table 6. Documents by Journal/Venue.

	Journal/Venue Name	Publisher	Contributions
Journal	*IEEE Access*	IEEE	8
	Future Generation Computer Systems	Elsevier	8
	Applied Sciences	MDPI	3
	IEEE Internet of Things Journal	IEEE	3
Conference	2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)	IEEE	2
	2021 IEEE Information Theory Workshop (ITW)	IEEE	2
Workshop	Business Process Management: Blockchain and Robotic Process Automation Forum	Springer International	5
	Financial Cryptography and Data Security. FC 2021 International Workshops	Springer Berlin Heidelberg	2

Then, excluding non-journal publications, we would have IEEE with 18 publications, followed by Elsevier with 15, MDPI with 13, and Springer with 5. This information is incredibly insightful when considering Table 7, which shows that only four journals published more than two papers on the subject. Conference venues and book sections, except for two venues, contributed no more than one document.

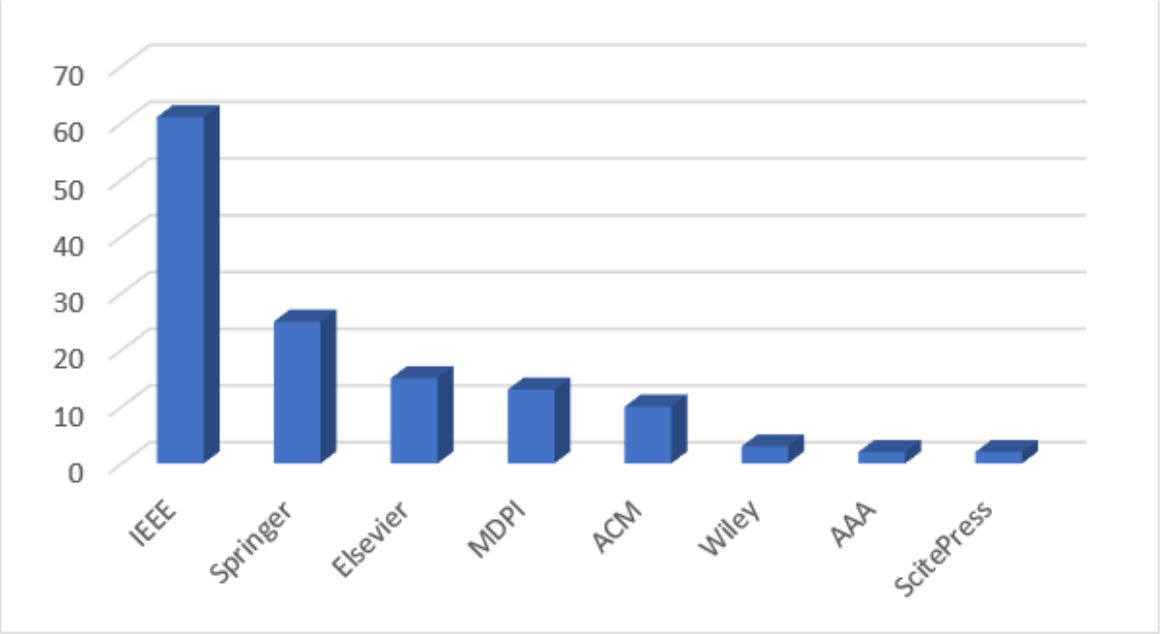


Figure 5. Documents by publisher.

Table 7. Distribution by category and article type.

Field		Article Type			Total
Main Categories	**Subcategories**	**Empirical**	**Theoretical**	**Review**	
Oracle Theory	Oracle Problem	6	6	6	18
	Proposal	19	3	0	22
	Architecture	18	8	9	35
Oracle Applied	Finance	16	2	4	22
	Data Management	9	5	0	14
	IoT *	9	2	1	12
	BPM *	4	3	1	8
	Supply Chain and Traceability	6	1	1	8
	AI *	4	3	0	7
	Cloud Computing	4	1	0	5
	Healthcare	4	0	0	4
	Transport	2	2	0	4
	Energy	2	1	0	3

* IoT = Internet of things; BPM = business process management; AI = artificial intelligence.

As shown in Table 6, the journals that published more contributions are *IEEE Access* and *Future Generation Computer Systems*, both with eight contributions. Among the other venues, the only notable is *Business Process Management: Blockchain and Robotic Process Automation Forum*, which contributed five book chapters.

4.4. Article Type, Fields, and Keywords

Table 7 provides an overview of the paper types determined by fields based on the main categories and sub-categories indicated in Section 3.1. It emerges as more than half (103); precisely, 63% are empirical papers, 23% are theoretical papers, and 14% reviews. At

the general level, the majority of academic research over oracles is of an empirical nature. Nevertheless, these data still need to be distinguished by field of research.

Concerning division by category, despite the higher number of sub-categories, the total number of papers belonging to OT (75) is slightly below those on OA (87). This is understandable, considering that oracles are still in their early-stage development, and a heterogeneity of views on how they should function and operate still exists. Although the majority of articles are still empirical, they are well balanced with theoretical and review types for the "architecture" and "oracle problem" sub-categories.

The second thing that emerges is that proposals are mainly of empirical/experimental nature, which bodes well for the birth of oracle frameworks in cooperation among or fully developed by academic institutions.

Regarding "oracle applied (OA)" papers being ideally a more practical area compared to OT, why an imbalance (except for BPM, AI, and transport) exists between empirical and theoretical papers is understandable. Furthermore, the smaller category size explains why only seven review papers were retrieved. By analyzing sub-categories, we can observe that some areas have fewer contributions than others. The finance sector leads with contributions, with 22 contributions, followed by data management (14) and IoT (12). Given the higher advancement level of blockchain applications in these sectors and the empirical nature of academic contributions, why other sectors, such as healthcare, transport, and energy, have less than five contributions is also understandable.

Keywords are also an important parameter to consider when evaluating a sample. A total of 650 keywords were extracted from the sample, which means a media of 3.9 per article. While some articles had six or more keywords, others (mainly preprints) had none. After duplicates and plurals were removed, 307 unique keywords were found. To avoid biases with the research strings used, however, we excluded keywords such as "blockchain" and "oracle(s)" from the analysis. Keywords composed of multiple words (e.g., Smart Contract) were considered unique, and those composed of banned keywords, such as "price-oracles", were not excluded. The choice to leave keywords composed of the two banned words lies in the idea that, while those keywords alone are common for all papers, composed keywords, such as centralized oracle or blockchain interoperability, are proper in specific sectors, which will benefit from homogeneous keyword usage. Plurals were also merged with singular forms (e.g., contract/contracts). Figure 6 shows the word cloud made with all the keywords in the sample. Notably, the most frequently used keywords are smart contract and Ethereum, with 67 and 21 occurrences, respectively. Whereas the keyword smart contract says very little about our sample, the recurrence of "Ethereum" surely reflects the most common study environment on oracles that appear to be the Ethereum network. Other keywords used are internet of things (8), consensus (7), and cryptocurrencies (5), whereas some have a lower currency rate. Interestingly, of the entire sample of 307 keywords, the majority (250) occurred only once. Keywords were also divided into categories to achieve good data breakdown.

After the most common keywords (e.g., Ethereum and Smart Contract) were excluded, excessive heterogeneity was still apparent, even after dividing them by categories. Composite keywords, such as "business process monitoring" and "business process management", were merged (e.g., business process) for consistency. In Table 8, keywords with higher occurrences divided by categories are listed. These data are useful for indexing purposes and for research to be easily retrieved by the appropriate audience. The "transport" category was excluded from the table because of excessive heterogeneity.

Figure 6. Keywords word cloud.

Table 8. Keywords by category.

Category	Keyword	Number	Category	Keyword	Number
Architecture	Architecture	3	AI	Artificial Intelligence	2
	Consensus	2		Machine Learning	2
	Data	3	BPM	Business Process	4
	Decentralized	3		Privacy	2
	Pattern	3		Service Composition	2
	Transaction	2	Cloud Computing	Cloud Computing	2
	Zero Knowledge Proof	2		Fog Computing	2
Finance	Cryptocurrencies	4	Data Management	5G	2
	Decentralized Finance	3		Certificate	2
	DeFi	3		Cross-Chain	2
	Gas	6		Data	5
	Financial	2	Energy	peer-to-peer	2
	Security	2	Healthcare	Healthcare	2
	Transaction-fees	2		Personal Health Records	2
IoT	Internet-of-things	5	Supply chain	Internet-of-things	3
	IoT	5		Supply chain management	3
Proposal	Consensus	3	Oracle Problem	Trust	3
	Decentralized Oracle	3		Real-World	2

4.5. Contribution by Author/Institution and Metric

The most cited papers, authors, and contributing institutions are displayed in Tables 9–11, respectively. Building on prior bibliometric analyses [62–65], the papers were ordered in terms of citations; therefore, the ten papers displayed in Table 10 are the most cited ones. However, institutions were ordered in terms of the papers produced. The list was not limited to ten but is restricted to those who provided at least three contributions. The most cited authors were selected with a mixed approach. Ordering authors by citation would have resulted in a biased list because of papers with many coauthors and citations. Therefore, to be inserted into the list, one requirement is that the author has to have produced at least two publications and has to be the first author for at least one of them. The requirement of at least two publications is to avoid the insertion of authors who have randomly contributed to a related paper. Then, assuming that the first author is the lead or the most contributing author, having first authored a paper appears to also be a necessary requirement. However, to also provide visibility to coauthors, Appendix B shows a list of coauthors who contributed to at least three papers. We were aware that a higher number of papers produced or a higher number of citations would not necessarily imply a higher impact or contribution in the field of oracle research. Such a claim would require a thorough study of academic contributions to the development of successful oracle applications, which is beyond the scope of a bibliometric analysis. In this research, a parameter, such as citations or produced papers, will correspond to a notable interest in the produced research of an author or a major effort from the institution to investigate the related field. The retrieved parameters do not reflect or question, in any case, the quality of an author's or institution's publication.

Table 9. Ten most cited papers.

#	Title	Author/s	Year	Cit *	Institution	I/T *
1	The Blockchain as a Software Connector	Xu, Xiwei; Pautasso, Cesare; Zhu, Liming; et al.	2016	524	UNSW Sidney	CP
2	Architecture for Blockchain Applications	Xu, Xiwei; Weber, Ingo; Staples, Mark	2019	180	UNSW Sidney	BS
3	Astraea: A Decentralized Blockchain Oracle	Adler, John; Berryhill, Ryan; Veneris, Andreas; et al.	2018	121	University of Toronto	CP
4	Blockchain for COVID-19: Review, Opportunities, and a Trusted Tracking System	Marbouh, Dounia; Abbasi, Tayaba; Maasmi, Fatema; et al.	2020	107	Khalifa University	JA
5	Trust management in a blockchain based fog computing platform with trustless smart oracles	Kochovski, Petar; Gec, Sandi; Stankovski, Vlado; et al.	2019	98	University of Ljubljana	JA
6	A Pattern Collection for Blockchain-based Applications	Xu, Xiwei; Pautasso, Cesare; Zhu, Liming; Lu, et al.	2018	80	UNSW Sidney	CP
7	Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges	Al-Breiki, Hamda; Rehman, Muhammad Habib Ur; Salah, Khaled; et al.	2020	77	Khalifa University	JA
8	Analysis of Data Management in Blockchain-Based Systems: From Architecture to Governance	Paik, Hye-Young; Xu, Xiwei; Bandara, H. M. N. Dilum; et al.	2019	73	UNSW Sidney	JA
9	TLS-N: Non-repudiation over TLS Enabling Ubiquitous Content Signing for Disintermediation	Ritzdorf, Hubert; Wüst, Karl; Gervais, Arthur; et al.	2018	58	ETH Zurich	JA
10	Blockchain for 5G: Opportunities and Challenges	Chaer, Abdulla; Salah, Khaled; Lima, Claudio; et al.	2019	55	Khalifa University	CP

* Cit = citations (Google Scholar); I/T = item type; CP = conference paper; JA = journal article; BS = book section.

Table 10. Twenty most-cited authors.

#	Name	Institution	Documents	Citations
1	Xu, Xiwei	UNSW, CSIRO-DATA61	6	908
2	Adler, John	University of Toronto	2	133
3	Lo, Sin Kuang	UNSW, CSIRO-DATA61	2	116
4	Kochovski, Petar	University of Ljubljana	2	115
5	Caldarelli Giulio	University of Verona	5	109
6	Omar Ilhaam A.	Khalifa University	2	107
7	Al-Breiki, Hamda	Khalifa University	2	97
8	Liu Xiaolong	Fujian Agriculture and Forest University	2	54
9	Carminati Barbara	University of Insubria	3	50
10	Rondanini Christian	University of Insubria	3	50
11	Battah, Ammar	Khalifa University	3	50
12	Madine, Mohammad Moussa	Khalifa University	3	49
13	Beniiche, Abdeljalil	INRS Montreal	2	44
14	Moudoud, Hajar	Sherbrook University	3	34
15	Tucci-Piergiovanni Sara	CEA-LIST	2	30
16	Di Ciccio, Claudio	Sapienza University of Rome	3	29
17	Ellul, Joshua	University of Malta	3	29
18	Merlini Marco	University of Toronto	2	26
19	Yeh, Lo-Yao	National Chi Nan University	2	18
20	Pierro, Giuseppe	University of Cagliari	2	17

Table 11. Most productive institutions.

#	Institution	Number	OT	OA
1	Khalifa University	13	1	12
2	University of Verona	8	5	3
3	UNSW, CSIRO-DATA61	6	5	1
4	University of Toronto	5	5	0
5	Beijing University	4	2	2
6	Technische Universität Berlin	3	1	2
7	University of Insubria	3	0	3
8	University of Ljubljana	3	1	2
9	University of Potsdam	3	0	3
10	INRS Montreal	3	1	2

As explained, information gathered with the above-mentioned approaches is provided in separate tables for clarity, but they should be discussed together to better grasp the meaning of the data.

The most cited author is Xu Xiwei (908 citations) from the University of New South Wales (UNSW) CSIRO-DATA61. She had co-authored the first two most-cited papers and four among the first ten. She started contributing to the subject in 2016, and given that her last paper on the topic was published in 2021, she appears to still be investigating the subject. All the included papers published by the UNSW are first-authored by her, except for one by Lo Sing Kuang, who is also among the most cited authors (116 citations). UNSW ranks third among the most productive institutions, with research mainly focused on oracles' architectures. The second most cited author is John Adler from the University of Toronto, who authored the third most cited paper (133 citations). In the University of Toronto, Merlini Marco is also among the most cited authors, and this institution is particularly focused on investigating decentralized oracle mechanisms. The sixth and seventh most cited authors are Omar Ilhaam A. and Al-Breiki Hamda from Khalifa University, with 107 and 97 citations, respectively. From the same university are also Battah, Ammar, and Madine, Mohammad Moussa, who are also among the most contributing authors but with fewer citations (50 and 49, respectively). Notably, Khalifa University is the most productive institution in the field, with 13 documents produced, of which 3 were among the ten most

cited and 4 were among the first twenty. Observing the coauthorship, apart from the four most cited first authors, many other authors from the same university also participated in the research studies. Among them, Muhammad Habib Ur Rehman and Davor Svetinovic are the most cited, with 144 citations each. These findings provide an idea of institutions that are heavily investing in this sector. Furthermore, this institution contributed at least one paper to every oracle application category (except for business process management (BPM) and energy). Furthermore, in addition to offering contributions to the healthcare and data management fields, they also produced research to address the oracle problem. Moreover, the University of Verona is also focused on addressing the oracle problem, which is ranked second by the number of articles produced.

However, publications from this institution are relatively recent and are not among the top-cited publications. From the same country (Italy), the University of Insubria is also among the most productive institutions, and two authors, Carminati Barbara and Rondanini Christian, are among the most cited (50 citations each). Studies from this university and its researchers were mainly concerned with OA as an IoT in business processes. Another notable institution is the University of Ljubljana, from which its contributions focus on cloud/fog computing and the oracle problem. The institution also belongs to the fifth most cited paper [66] and the fourth most contributing author, Petar Kochovsky, with 115 citations.

Among the most productive institutions, five other institutions emerged, for which their researchers were also among the most impactful ones. These institutions include Beijing University, Technische Universität Berlin, the University of Potsdam, and the Institut national de la recherche scientifique (INRS) of Montreal. Beniiche, Abdeljalil, from INRS of Montreal, is the most cited in this group (44 citations), and his main contributions focused on OT. Finally, from Technische Universität Berlin is Ingo Weber, which, although not the first author of any of the papers in the sample, has coauthored some of the most cited ones (303 total citations).

5. Converging Studies, Research Themes, and Research Directions

This section of the paper is dedicated to reviewing and discussing the collected studies, with the aim of extracting critical features concerning the related fields and the research direction. The objective is to understand which aspects of oracles have been investigated, which methods are used, and what results have been generated to highlight emerging research trends. Furthermore, by comparing research papers, converging studies are highlighted to promote cooperation between institutions. Appendix C also provides a complete list of papers sorted by institutions and categories to better understand the research distribution.

5.1. Oracle Theory

Subjects pertaining to the OT and the oracle architecture comprise many different studies. Given that oracles are a niche area of investigation, they are not officially classified by type, and their characteristics have yet to be defined. A group of studies has been dedicated to investigating common patterns that emerge from oracle architectures, with the aim of classification and improvement [67–71]. Pasdar et al. [67] differentiated reputation-based and voting-based oracles, explaining how each design provides the answer to the smart contract. Muhlberger et al. [68] instead distinguished oracles between inbound and outbound, depending on the direction of the data flow (push and pull). Inbound oracles provide data to the blockchain, whereas outbound ones transfer data from the blockchain to the real world. Specific examples are also made of blockchain applications, where data are pushed or pulled into the smart contract. Xu et al. and Mammadzada et al. built a framework to select the most appropriate oracle design (in terms of security and data management) according to different blockchain applications [32,69,71].

Other works from the University of Colorado [72], Jiamusi University [73], and Hong Kong University [74] focused on the security and privacy challenges of Oracle-based smart

contracts. Their research study mainly examined how to identify and prevent oracle malfunction (integrity), guarantee that data collected is exploited solely by the smart contract (confidentiality), and prevent downtime or censorship attempts (availability). A group of works from Montana State University [75], University of Sfax [76], and the University of Cagliari [77] focused instead on oracle fees and gas-price oracle malfunctions. The work of Montana State University investigated the reasons that led to gas price oracle failures, and the study from the University of Cagliari outlined the failure rate of gas price oracles with an empirical approach. The paper from the University of Sfax compared different gas-pricing techniques with the aim of improving oracle reliability.

Another central subject in OT is the oracle problem issue, for which many contributions were retrieved. A group of papers focused on explaining the oracle problem, whereas others focused more on empirically investigating the subject to overcome the issue. Two papers from the University of Ljubljana and Max-Planck Institute introduced the oracle problem from a legal point of view [29,30]. In this paper, the oracle's role as legal actors and their responsibility as a trusted entity were investigated. A similar discussion can also be retrieved in Mezquita et al. [78], which, however, focused on the legal audit of smart contracts. A thorough discussion of the audit of the smart contract in light of the oracle problem could instead be found in two works by Mark Sheldon D. [13,79], which first introduced the problem of auditing contracts and then offered insights for future auditors to perform the task better.

Other papers from the University of Verona and Khalifa University focused on investigating trust models and the consequences of having untrusted oracles in various sectors, such as Intellectual Property Rights (IPRs), DeFi, and supply chain [6,9]. Considering the amount of money managed by DeFi platforms, the financial implications are alarming [14,46]. Singapore University has also confirmed this result with research focused on investigating the reliability of oracles in DeFi applications [1]. Finally, studies from the Chiba University of Technology and the University of Dallas explored, with empirical data, the incentives of oracles to cheat or fail to transmit information [80,81]. The focus of these studies has mainly concerned the issue of how trust can be built or undermined in digital economies and how collective intelligence helps prevent selfish individuals from performing disruptive actions in the community.

The last subject of OT pertains to oracle proposals. Proposals concern elaboration from scratch or improvements of oracle trust models, such as the one discussed by Al-Breiki et al. [6]. However, because of excessive heterogeneity, finding research themes and research directions was not feasible for this category. Furthermore, proposals were retrieved in a balanced distribution among institutions in various countries, apart from being heterogeneous. Therefore, a considerable convergence of studies among institutions could not be retrieved. Table 12 summarizes the content of the paragraph.

Table 12. Oracle theory: Themes, directions, and converging studies.

Research Themes	Research Directions	Converging Studies
Oracle Architecture		
Oracle pattern	- Find a univocal Oracle Taxonomy - Distinguish between oracle types and features	Macquarie University [67] Vienna University [68]

Table 12. *Cont.*

Research Themes	Research Directions	Converging Studies
Oracle Architecture		
Oracle privacy and security	- Identify and prevent oracle malfunction - Ensure data confidentiality - Guarantee censorship resistance and limit downtime periods	University of Colorado [72] Jiamusi University [73] Hong Kong University [74]
Oracle pricing and fees	- Oracle pricing techniques - Gas pricing reliability	Montana State University [75] University of Sfax [76] University of Cagliari [77]
Oracle Problem		
Oracle as a legal actor	- Define the relationship between oracles and other parties - Define oracles' responsibilities	University of Ljubljana [29] Max-Planck Institute [30]
Oracle auditing	- Define the means to detect improperly designed oracles	University of Salamanca [78] John Carroll University [13,79]
Trust model	- Adapt a trust model design to specific blockchain applications	University of Verona [9] Khalifa University [6]
Incentive to cheat	- Define the nature and driver of trust in digital economies - Develop means to prevent selfish behavior	Chiba University of Technology [81] University of Dallas [80]

5.2. Oracle Applied

Oracle-applied research is focused on various sectors. As expected, because of the resonance and hype that cryptocurrencies attract, finance applications constitute the widest sample. Although multiple institutions have investigated the subject, they show similarities in their focus. Studies from Concordia University, the University of Houston, and the University of Singapore focused on the very role of DeFi oracles: how they work, how they are designed, and how they interact with the underlying blockchain. Existing oracle types are also compared (in terms of efficiency), analyzing how data are retrieved, aggregated, and pushed into the blockchain [1,48]. Kaleem and Shi [82] also provided an overview of the percentage of DeFi oracle calls over total oracle calls. They discovered that almost 75% of ChainLink oracle calls are from Synthetics, a derivative-based DeFi project. Other studies from the University of Verona and Delhi Technological University focus on known threats to DeFi oracles. Although some, such as technical malfunctions or Sybil attacks, are efficiently spotted and addressed, others, such as frontrunning or flash loans, are still difficult to prevent and sometimes even to spot [46,83]. Three studies from Concordia University, Delhi University, and Delft University of Technology focused on the role of the oracle as a means to manipulate the market, showing the possible risks connected with its use and misuse [48,83,84]. Whereas the first two have a more theoretical slant, the third one with an empirical approach investigates how arbitrageurs exploit oracle vulnerabilities.

Empirical research from the Oxford-Hainan Blockchain Research Institute, Singapore University of Technology and Design, and Delft University of Technology further contributed to this field of study. The first proposed BLOCKEYE, a device able to hunt attacks on DeFi and oracle manipulations, for which the research team had already presented some experimental results [85]. Using primary data, the second showed the deviance rates of four oracle services to enlighten the oracle's reliability and possible malfunctions [1]. Finally, the third investigates how arbitrageurs' activities can influence or manipulate price oracle data feeds [84].

Another group of studies discussed how oracles intervene according to a specific financial application (e.g., loans, trading platforms, trust services) [46,86]; however, apart from two papers from Khalifa University and the University of Clermont Auvergne, which both investigated e-auctions, the rest had heterogeneous aims. Both studies on e-auction had an experimental approach and proposed a new auction service based on the Ethereum blockchain, specifying the role of oracles and how to overcome possible security issues [87,88]. Three studies also investigated the role of oracles in cross-chain asset transfers. Whereas the study from the University of Lisbon provides an overview of different cross-chain techniques, the work from the University of Verona discusses the utilities of the transferred tokens based on their provenance [51,89]. Another study from Beihang University proposes PracticalAgentChain, an intermediary between the data oracle and provenance blockchains (e.g., Bitcoin and Ethereum) [90]. The system works as a reputation-based trading pool and utilizes Town Crier for reliable oracle service.

Business process management research is mainly built toward the ability of blockchains to monitor business processes efficiently. Di Ciccio et al. [91] provided an overview of how the monitoring business process with blockchain can be achieved and discussed the challenges eventually faced. An extensive description of oracle implications is provided, first by discussing how oracles should be synchronized (time management) to avoid delay of reports. Second, the reliability of oracles is discussed to ensure that the data are not manipulated. Third, the flexibility of oracles should be guaranteed so that the smart contract can select the best data source according to the monitored event. Fourth, blockchain data are aligned with real-world data so that the event sequence is not misrepresented. Concerning the timeliness and alignment of oracles, a group of works by the University of Potsdam [92–94] proposed a "deferred choice pattern", given that the time of transaction is not known in advance. Their model involves an extended oracle architecture to make all historical process data available (history oracle), sensitive to any unexpected change (publish–subscribe oracle), and preserve privacy and data efficiency. The last is achieved by performing part of the computation and variable evaluation off-chain (conditional oracle variants). More focused on the privacy of business process execution, works from the University of Insubria have proposed an encryption mechanism to ensure data confidentiality, even in the presence of an untrusted oracle. The works also verify the encryption data consequences on smart contracts and transaction overhead [95,96].

As for the supply chain and traceability fields, the works seem heterogeneous, although eight entries were retrieved. Construction, fashion, and food supply chains were investigated, as well as the traceability of vehicles and COVID-19 infections [97–99]. Sanchez-Gomez et al. proposed that, for a blockchain traceability solution to work, it must operate on a dedicated layer/network, with traceability data separated from the blockchain data in which a reliable data verification mechanism should be implemented. Oracles and external APIs, in their design, play a critical role [100]. Following this approach, Moudoud et al. [99] proposed collecting traceability data on a cloud, where a network of trusted oracles (recognized by the signature) approves the most reliable information. Concerning the reliability of oracles, a study by Marbouh et al. [101] proposed rules to evaluate an oracle's reputation in tracing COVID-19 cases. Thresholds also determine the oracles' inclusion or exclusion from the trusted network.

Focusing on the construction supply chain, Lu et al. [97] proposed the use of smart construction objects (SCOs) as blockchain oracles given their intrinsic characteristics. SCOs are, in fact, able to sense the surrounding environment and efficiently communicate the information acquired. Lastly, they have the autonomy to respond to certain situations based on predefined rules. Victor and Zickau [98] proposed network operator companies as tracking oracles given the massive presence of cellular radio towers. Finally, Powell et al. [102] discussed the issue of attaching a physical product to the blockchain in order for it to be traced by an oracle. Studies from the University of Verona have also investigated this specific aspect [8,103]. All of these studies support the idea that a known oracle identity is fundamental to achieving this task.

For healthcare, only four papers were retrieved and were all focused on the security and access control of patients' records [104,105]. Madine et al. [105] proposed a decentralized reputation-governed trusted oracle network for patient records to promote competition among oracles and ensure quick and reliable data transmission. However, they also proposed that, because of the sensitive nature of data transmitted, oracles should be approved by a regulatory agency. In a subsequent study in which they proposed a system of tokens to ensure patients' control of their medical records, they also debated the necessity of having a second oracle type for a time-based trigger events [104]. Research by Goncalves et al. [106] focused on the same objective but proposed a specific oracle solution with the Chainlink oracle provider and Ethereum blockchain.

Seven entries regarding applications in AI were retrieved. The central focus was to exploit automation and oracles to guarantee trust in data gathering and processing. As in the original idea of the software oracle problem, the objective was to reduce external parties' intervention in automated procedures. Studies from Toulouse University and INRS Montreal investigated AI-based oracles to provide non-forged results [107,108]. While the first aimed to complete the automation of the oracle ecosystem, the second proposed a more hybrid system between humans and machines. In particular, Beniice et al. [107] demonstrated that the presence of a third party, a human or social robot, plays an important role in a blockchain-enabled trust game. The works of El Fezzazi et al. [109] and Richard et al. [110] aimed to exploit blockchain oracle features to improve machine learning processes and predictive models to reduce dependency on third-party data feeds. Both offered a theoretical overview of the blockchain implementation outcome at the concept stage.

The IoT sector has 12 publications, and the main issue of investigation is the problem faced while ensuring that the data gathered by IoT devices are trustworthy and private. Gordon [111] and Vari-Kakas et al. [112] outlined the problem of secure and trustworthy data provenance within IoT systems. The first focuses on the problem of the authentication of IoT oracles on blockchains to ensure that data are submitted only by trusted oracles. It proposes that oracles submit their addresses along with data so that blockchain applications can easily verify data provenance. The second is focused on the statistical probability for an IoT oracle to deliver reliable data to the blockchain. In response to this issue, Shi et al. [113] proposed a secure and lightweight triple-trusted architecture to guarantee the unforgeability of data collected by trusted oracles. In their research, however, the premise is that oracles are trustworthy in the first place. By contrast, contributions from Khalifa University and Insubria University approached the confidentiality of the IoT. The first proposes implementing cloud computing and different access privileges to guarantee against unwanted data leakage. The second proposes an encryption model in which IoT and related data are only accessible by the intended users [7,114]. Whereas the first is more oriented toward the technical feasibility of IoT-based blockchain data gathering, Moudoud et al. [115] proposed an ad hoc blockchain architecture based on sharding and a peer-to-peer oracle network in order to manage IoT devices. Although at an early stage, the prototype already shows some experimental results.

As for cloud computing, only five studies were retrieved. Two were published at the University of Ljubljana and focused on how oracles can enhance trust and efficiency in a cloud computing platform. Defining the drivers of trust, a trust management scheme is proposed to show how a trusted data flow can be achieved between application components (e.g., camera, fog node, and cloud storage) [66]. In subsequent work, the research is extended, showing how oracles can increase scalability and cost-efficiency in federated edge-to-cloud computing environments by allowing transactions to be executed off-chain [116]. Tao and Hafid [117] also proposed introducing a computing oracle to reduce on-chain network usage. In line with these studies, Taghavi et al. [118] proposed oracles as a monitoring service for service-level-agreement violations in cloud environments. Utilizing a Stackelberg differential game, they also investigated the perfect balance between quality verification requests and monitoring prices.

A consistent group of papers applied oracles for data management. Comuzzi et al. [119] investigated how oracles impact data quality in terms of the timeliness, costs, and availability of data. They showed that availability increases by querying an external oracle service, but so do also costs. Battah et al. [120] proposed a reputation system to reward better-performing oracles to improve accessibility and costs, eventually increasing data quality. In a subsequent study, the authors better specified the drivers to discover trusted and better-performing oracles, also showing simulation results [121].

Other authors have focused on data communication between blockchains. Mitra et al. [122] proposed DE-PEG, a modification of the PEG algorithm said to reduce the cost of data availability oracles and thought to also prevent stalling attacks. Gao et al. [123] explored active communication between blockchains through oracles, specifying which type of data should be transmitted. However, in their research, they assumed that the oracle is always trusted, so they also did not provide a scheme to prevent oracle data manipulation. Finally, Ouyang [124] proposed HBRO, an oracle system that enables communication between permissioned and permissionless blockchains concerning digital rights management (DRM). Once DRMs are elaborated on the permissioned chain, they are securely transmitted through the data oracle to a permissionless blockchain with a notary mechanism.

Works in the transport sector have focused on the security, privacy [125,126], and efficient identification of vehicles [127], as well as data processing for intensive transport environments, such as commercial waterways [128]. Whereas the works from Khalifa University and Guangxi University discussed the implementation of Chainlink for autonomous vehicle test-case repositories and identification, the works from other universities proposed their own oracle design for efficient data transmission in the transport industry.

Finally, three entries were retrieved for the energy sector. All three were published between late 2021 and early 2022. The shared vision aims at decentralizing the energy market, but with a different focus. Antal et al. [129] proposed an energy flexibility token to incentivize renewable energy production at the local level. Zeiselmair et al. [130] implemented a decentralized oracle system and zkSNARKs to improve renewable energy certificate allocation. Lastly, Weixian et al. [131] investigated efficient oracle designs to guarantee secure and unforgeable data transmission between actors involved in the energy market. An overview of the above-mentioned results can be found in Table 13.

Table 13. Oracle applied: Themes, directions, and converging studies.

Research Themes	Research Directions	Converging Studies
	Finance	
DeFi Oracle	- Functioning of DeFi oracles. - Compare the efficiency of existing DeFi oracles. - Analyze the DeFi oracles network usage	Concordia University [48] University of Singapore [1] University of Houston [82]
DeFi oracle manipulation	- Common attack vectors - Oracle defense mechanisms - Evaluate arbitrageurs' influence over oracle price feeds	University of Singapore [1] University of Verona [46] Delhi Technological University [83] Oxford-Hainan Blockchain Research Institute [85]
Defi applications in cross-chain transactions.	- DeFi applications and specific oracle designs - Cross-chain cryptocurrency transfer data management	China Merchant Group [86] University of Verona [46,51] University of Lisbon [89] Beihang University [90] Khalifa University [87] Toulouse University [88]

Table 13. *Cont.*

Research Themes	Research Directions	Converging Studies
Finance		
Business Process Management		
Business process monitoring	- Auditing of processes - Timeliness of execution - Privacy and data efficiency	Sapienza University of rome [91] University of Potsdam [92–94] University of Insubria [95,96]
Supply Chain Management		
Supply chain oracles	- Define the most appropriate oracle for supply chains - Define rules to establish trusted oracle inclusion/exclusion	Khalifa University [101] University of Hong Kong [97] Technische Universit¨at Berlin [98]
Supply chain data transmission.	- Define appropriate traceability data storage platforms - Discuss the attachment vector between the physical product and the blockchain	Universit'e de Sherbrooke [99] University of Seville [100] Queensland University of Technology [102] University of Verona [8]
Healthcare		
Patient records	- Healthcare oracle management - Oracle tasks in the healthcare sector	Khalifa University [104,105] University of Antwerp [106]
Artificial Intelligence		
AI-based oracles	- Improve oracle efficiency through automation	Toulouse University [108] Montreal University [107]
Oracles and machine learning	- Reduce dependency on oracles exploiting predictive models	Sidi Mohamed Ben Abdellah University [109] Bina Nusantara University [110]
Internet of Things		
IoT as data oracles	- Ensure that data collected by IoT sensor is not forged - Ensure that only intended IoT devices are allowed to upload data on the ledger	Saint Mary's University [111] University of Oradea [112] National University of Defense Technology [113]
IoT confidentiality	- Ensure that IoT data are not leaked - Allow IoT data to be accessed only by the intended users	Khalifa University [7] University of Insubria [114]
Cloud Computing		
Application management	- Oracles as a means to reduce transaction costs and increase scalability - Oracles for Service Level Agreement management and monitoring	University of Ljubljana [66,116] Khalifa University [118] University of Montreal [117]
Data Management		
Data privacy and quality	- Balance availability and costs - How to prevent unwanted data access	Polytechnic University of Milan [119] Khalifa University [120]
Cross-chain communication	- How to prevent stalling attacks - Discuss allowed data type	University of California [122] Jinan University [124] Beijing University [123]

Table 13. *Cont.*

Research Themes	Research Directions	Converging Studies
Finance		
Transport		
Vehicle management	- Guarantee privacy in the vehicle identification - Process autonomous vehicle data	Khalifa University [125] Guangxi University [127]
Energy		
Energy market management	- Incentivize renewable energy production - Improve privacy in the energy market	Technical University of Cluj-Napoca [129] Technical University of Munich [130]

6. Discussion

The literature review showed some interesting insights about themes covered by the existing studies and areas that require more attention. Although a group of studies have tried to classify oracles, the very concept of oracle is still not clearly defined. Oracles are generally identified as data-feeding ecosystems, which can come in various forms or structures, but if this is the purpose of oracles, then anything that provides data for a blockchain is an oracle. Therefore, rollups or bridges should also be considered oracles. We could argue, for example, that the lighting network is an oracle for the bitcoin network, but then we may also have oracles on the lighting network. Therefore, the boundaries of what can be defined as an oracle should be clearly settled.

The classification of oracles is also quite heterogeneous, and it often reduces the clarity of the presented research. In addition to software and hardware, we also have reverse, centralized, decentralized, computation, consensus, and voting-based oracles. Often, they refer to the same type with different names or to different types with the same name (e.g., decentralized and consensus-based oracles). Similarly, the issue of having trusted entities in trustless environments, often referred to as the "oracle problem", has also been labeled in some research as the "oracle paradox". Likewise, the coexistence of inbound and outbound oracles is referred to equally as the "dual-oracle problem" or "dual simplex communication". Therefore, efforts should be made in this regard so that research can build on a common oracle taxonomy.

Concerning the investigation of oracle trust models, this seems to still be a niche field, and despite the few contributions, the very concept of trust lacks a broader discussion and clarification. Indecisiveness on whether an oracle should be trustless or "provably honest" is apparent, which is, in theory, not really the same concept.

With regard to oracle proposals, they are heterogeneous, but almost all focus on decentralized or consensus-based oracle systems. Centralized oracles, such as the ones proposed in [124,132,133], should also be worthy of investigation. For certain data types that are not in the public domain and where data sources are limited, a centralized and secure data channel would be more appropriate than a slow, expensive, and probably less secure decentralized oracle type. A trusted oracle is indeed a single point of failure, but it is undoubtedly more efficient as long as it resists attacks and behaves honestly. Therefore, more research is expected and needed in that direction. Furthermore, despite the plethora of emerging oracle solutions on the market, on the academic side, the trend is to propose a new oracle solution or utilize only the most known or advertised ones (e.g., Chainlink or Provable). Therefore, an exploration of emerging oracle solutions by academic researchers is expected and required.

Another issue that emerged in this study is that studies concerning oracle solutions for DRM, such as the one by Ouyang [124], are limited. As the original idea of blockchain proposed by Haber and Stornetta [55] was to authenticate digital documents, more attention to this sector was also expected in blockchain oracle research.

Finally, concerning a well-known issue discussed in 2018 by Song [134] (that is, how to link a physical object to the blockchain), an effective method to fill this gap has not been developed despite the plethora of research and proposals. This considerable limitation greatly undermines the feasibility of blockchain-based proposals and applications, especially for the traceability sector (but also in healthcare or BPM). Therefore, more than speculating on the hypothetical advantages of having a blockchain-based tracing system, a considerable effort should be made to understand whether a physical product can be "attached" to the blockchain in the first place. Building on the above-mentioned limitations, Table 14 suggests some research themes along with their expected/desired outcomes.

Table 14. Suggested research themes.

Field	Theme/s	Expected/Desired Outcomes
Oracle theory	- Contribute to a univocal oracle taxonomy	- Make oracle literature more accessible to a broader audience - Attract more research in the oracle field
	- Define the boundaries of oracles' definition	- Reduce heterogeneity within oracle literature - Improve consistency of research
	- Define the theoretical background to which the "trust" discussed in oracle literature should adhere	- Develop more robust trust models - Reduce the heterogeneity of proposals
	- Investigate resistant and trusted centralized oracle types	- Adapt oracles to specific data types with limited sources.
Oracle applied	- Integrate oracles in digital rights management	- Broaden blockchain-based intellectual property use cases.
	- Investigate robust links between physical products and the blockchain	- Improve any blockchain-based application involving physical products (e.g., traceability).
	- Investigate emerging oracle solutions	- Promote active collaborations between academia and oracle providers

7. Conclusions

This paper undertook a bibliometric analysis of published studies about blockchain oracles. The aim was to display publication trends along with preferred outlets and publishers. The most-cited papers and authors and the most contributing institutions are also shown. After the selected literature was reviewed, emerging themes, research directions, and converging studies were discussed to promote innovation and cooperation between institutions.

The obtained results show that, within seven years of academic production, only 162 papers (including non-peer-reviewed) were retrieved in scholarly databases. This result supports the view that blockchain oracle is still a widely neglected subject, despite its crucial importance. The review also reveals heterogeneity in the oracle literature; therefore, major efforts are required to find a widely accepted oracle taxonomy. Furthermore, limited oracle selection suggests the need for more active collaboration between practitioners and academia. Finally, more theoretical work is required on the underlying trust concept that identifies oracles as "trusted" ones.

The findings of this research are useful for academics, students, and practitioners. Offering an overview of institutions investigating a specific field, this study can promote cooperation between existing or entering research teams in the blockchain oracle domain. This, in turn, can constitute a reference for entrepreneurs undertaking blockchain-based projects. Students and other academics can then utilize a resource on the state-of-the-

art knowledge of related fields and investigate emerging gaps (e.g., missing resource management contributions) or create other research by building on existing studies.

This paper also has limitations given the scarcity of retrieved material that determined low numbers in absolute terms in all the tables and figures. As specified in Section 3, a degree of subjectivity in the presented results cannot be excluded. Whereas previous studies inspired the method and bibliometric research, the author had to select them arbitrarily. Subjectivity can also be found in the sample classification, given that the division of topics into categories and sub-categories had to be performed manually. Again, the author wishes to reiterate that the data provided in this paper should not, in any case, be interpreted as a quality evaluation of the cited works. Because of the selection criteria, some works or authors may have been excluded inadvertently. Further studies can build on this bibliometric analysis to investigate the trust models adopted and presented in the published literature and the preferred oracle applications for academic investigations.

Funding: This research was funded by "Emma Gianesini Fund", managed by UniCredit Foundation.

Data Availability Statement: Almost all the data generated or analyzed during this study are included in this published article. Unreported data are available upon request.

Conflicts of Interest: The author declares no conflict of interest.

Appendix A

Table A1. Relevant contribution example.

Paper Title	Oracle Contribution	Reference
The limits of smart contracts	Provides an analysis of the role of oracle from a legal point of view	[30]
LoC—a new financial loan management system based on smart contracts	Discusses how oracles can be implemented to ensure data privacy in loan management	[135]
A pattern collection for blockchain-based applications	Describes different oracle types and how to recognize the most suitable one according to the needs	[69]
On the characterization of blockchain consensus under incentives	Compares blockchain consensus and oracle consensus under specific incentive mechanisms	[136]
Distributed network slicing management using blockchains in e-health environments	Shows the implementation of a decentralized oracle solution for the management of patient records	[106]
Blockchain for COVID-19: review, opportunities, and a trusted tracking system	Outlines a means to recognize a trusted oracle network for tracking purposes	[101]
To chain or not to chain: a reinforcement learning approach for blockchain-enabled IoT monitoring applications	Presents a blueprint of a private network in which oracle contracts improve their efficiency according to data collected by IoT sensors	[137]
Blockchain as a platform for secure inter-organizational business processes	Discusses oracle data correctness and confidentiality in business process management	[95]

Appendix B

Table A2. Notable coauthors.

Full Name	Institution	Citations	Documents
Zhu, Liming	UNSW, CSIRO-DATA61	612	3
Ingo, Weber	Tu-Berlin	303	5
Jayaraman, Raja	Khalifa University	179	5
Veneris, Andreas	University of Toronto	160	5
Berryhill, Ryan	University of Toronto	147	3
Veira, Neil	SoundHound Toronto	147	3
Muhammad Habib Ur Rehman	Khalifa University	144	3
Davor Svetinovic	Khalifa University	144	3
Ellaham, Samer	Khalifa University	142	3
Yaqoob, Ibrar	Khalifa University	66	4
Ferrari, Elena	University of Insubria	50	3
Weske, Mathias	University of Potsdam	14	3

Appendix C

Table A3. Complete list of articles sorted by categories.

Oracle Architecture	Entries
Beihang University	[138]
CEA LIST Paris-Saclay University	[136,139]
Eindhoven University of Technology	[140]
ETH Zurich	[141]
Fujian Agriculture and Forest University	[31,142]
TU-Berlin	[143]
INRS, Montr'eal	[33]
IST Austria	[144]
Jinan University	[145]
Langfang National University	[146]
Macquarie University	[67]
Nirma University	[147]
RMIT University	[148]
Hong Kong University	[74]
University of Canterbury	[149]
University of Dallas	[80]
University of Salamanca	[78]
University of Tartu	[32]
University of Toronto	[150]
UNSW Sidney CSIRO DATA61	[60,69–71,151]
Vienna University of Economics and Business	[68]
Jiamusi University	[73]
Sapienza University of Rome	[152]
University of Colorado	[72]
Carnegie Mellon University	[153]
University of Malta	[154]
Politecnico di Milano	[155]
Montana State University	[75]
University of Rijeka	[156]

Table A3. *Cont.*

Oracle Architecture	Entries
Oracle Problem	**Entries**
Chiba, Institute of Technology	[81]
EBS University	[3]
IST Austria Klosterneuburg	[157]
John Caroll University	[13,79]
Khalifa University	[6]
Max Planck Institute	[30]
Montclair state University	[4]
Technical University Munich	[158]
University of Applied Sciences Offenburg	[159]
University of Ljubljana	[29]
University of Verona	[2,8,9,14,38]
Imperial College London	[160]
University of Connecticut	[161]
Oracle Proposal	**Entries**
Beijing University of Technology	[162,163]
Chungnam National University	[132]
Delft University of Technology	[164]
Kleros Cooperative	[165]
Kyushu University	[166]
National Taiwan University	[167]
Sogang University	[168]
Swiss Federal Institute of Technology	[169]
Technische Universit¨at Berlin	[170]
University of Illinois	[171]
University of Toronto	[172–175]
Hamburg University of Technology	[176]
South Asian University	[177]
Dublin City University	[178]
South China Normal University	[179]
Rensselaer Polytechnic Institute	[180]
University of Applied Sciences—Kufstein	[181]
Shanghai Jiao Tong University	[133]
Finance	**Entries**
Aalborg University Copenhagen	[182]
China Merchants Group	[86]
Concordia University Montreal	[48]
Cornell University	[61]
Delhi Technological University	[83]
Khalifa University	[87]
Nanjing University	[183]
NTNU Norway	[135]
Oxford-Hainan Blockchain Research Institute	[85]
SUTD Singapore	[1]
Universit'e Clermont Auvergne	[88]
University of Cagliari	[77,184]
University of London	[185]
University of Potsdam	[94]
University of Sfax	[76]
University of Houston	[82]
University of Luxembourg	[186]
Beihang University	[90]
University of Verona	[46,51]
Delft University of Technology	[84]

Table A3. *Cont.*

Oracle Architecture	Entries
Artificial Intelligence	**Entries**
Bina Nusantara University	[110]
INRS Montreal	[107]
Khalifa University	[187]
Universidade da Beira Interior	[188]
University of Luxembourg	[189]
University of Toulouse	[108]
Sidi Mohamed Ben Abdellah University	[109]
Business Process Management	**Entries**
KAUST	[190]
University of Insubria	[95,96]
University of L'Aquila	[191]
University of Postdam	[92,93]
UEST—China	[192]
Sapienza University of Rome	[91]
Cloud Computing	**Entries**
Khalifa University	[118]
University of Ljubljana	[66,116]
Université de Montréal	[117]
North Carolina State University	[193]
Data Management	**Entries**
Beijing University	[123,194]
Chiba Institute of Technology	[195]
Kaunas University of Technology	[196]
Khalifa University	[120,121,197]
Rennes University	[198]
Shenzhen Technology University	[199]
UNIST—South Korea	[119]
UNSW Sidney CSIRO DATA61	[200]
University of Sherbrooke	[201]
University of California	[122]
Jinan University	[124]
Energy	**Entries**
Technical University of Munich	[130]
Technical University of Cluj-Napoca	[129]
Electric Power Research Institute—Beijing, China	[131]
Healthcare	**Entries**
Khalifa University	[101,104,105]
University of Antwerp	[106]
Internet of Things	**Entries**
Khalifa University	[7]
National Chi Nan University	[202]
NUDT—China	[113]
Qatar University	[137]
Saint Mary's University	[111]
Technische Universit¨at Berlin	[203]
University of Insubria	[114]
INRS, Montr'eal	[204]
Wayne State University	[205]
University of Oradea	[112]
Blockchain 5.0 OÜ	[206]
University of Sherbrooke	[115]

Table A3. *Cont.*

Oracle Architecture	Entries
Supply Chain and Traceability	**Entries**
University of Sherbrooke	[99]
Carlo Cattaneo University	[207]
Khalifa University	[208]
Technische Universit¨at Berlin	[98]
University of Hong Kong	[97]
University of Seville	[100]
University of Verona	[103]
Queensland University of Technology	[102]
Transport	**Entries**
Guangxi University	[127]
National Taiwan University	[126]
Khalifa University	[125]
Fraunhofer FIT and RWTH Aachen University	[128]

References

1. Liu, B.; Szalachowski, P.; Zhou, J. A First Look into DeFi Oracles. *arXiv* **2020**, arXiv:2005.04377.
2. Caldarelli, G. Real-world blockchain applications under the lens of the oracle problem. A systematic literature review. In Proceedings of the 2020 IEEE International Conference on Technology Management, Operations and Decisions, ICTMOD 2020, Marrakech, Morocco, 25–27 November 2020; pp. 1–6.
3. Egberts, A. The Oracle Problem—An Analysis of how Blockchain Oracles Undermine the Advantages of Decentralized Ledger Systems. Available online: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3382343 (accessed on 4 June 2022).
4. Albizri, A.; Appelbaum, D. Trust but Verify: The Oracle Paradox of Blockchain Smart ContractsTrust but Verify: The Oracle Paradoxof Smart Contracts. *J. Inf. Syst.* **2021**, *35*, 1–16. [CrossRef]
5. Antonopoulos, A.M.; Woods, G. *Mastering Ethereum—Building Smart Contracts and DAPPS*; O'Reilly: Tokyo, Japan, 2018.
6. Al-Breiki, H.; Rehman, M.H.U.; Salah, K.; Svetinovic, D. Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges. *IEEE Access* **2020**, *8*, 85675–85685. [CrossRef]
7. Al Breiki, H.; Al Qassem, L.; Salah, K.; Habib Ur Rehman, M.; Sevtinovic, D. Decentralized access control for IoT data using blockchain and trusted oracles. In Proceedings of the Proceedings—IEEE International Conference on Industrial Internet Cloud, ICII 2019, Orlando, FL, USA, 11–12 November 2019; pp. 248–257.
8. Caldarelli, G.; Rossignoli, C.; Zardini, A. Overcoming the blockchain oracle problem in the traceability of non-fungible products. *Sustainability* **2020**, *12*, 2391. [CrossRef]
9. Caldarelli, G. Formalizing Oracle Trust Models for blockchain-based business applications. An example from the supply chain sector. *arXiv* **2022**, arXiv:2202.13930. [CrossRef]
10. Lizcano, D.; Lara, J.A.; White, B.; Aljawarneh, S. Blockchain-based approach to create a model of trust in open and ubiquitous higher education. *J. Comput. High. Educ.* **2020**, *32*, 109–134. [CrossRef]
11. Gruner, A.; Muhle, A.; Gayvoronskaya, T.; Meinel, C. A Quantifiable Trust Model for Blockchain-Based Identity Management. In Proceedings of the 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 30 July–3 August 2018; pp. 1475–1482.
12. Caldarelli, G. *Blockchain Oracles and the Oracle Problem: A Practical Handbook to Discover the World of Blockchain, Smart Contracts, and Oracles—Exploring the Limits of Trust Decentralization*, 1st ed.; Amazon Publishing: Naples, Italy, 2021; ISBN 979-1220083386.
13. Sheldon, M.D. Auditing the blockchain oracle problem. *J. Inf. Syst.* **2021**, *35*, 121–133. [CrossRef]
14. Caldarelli, G. Understanding the Blockchain Oracle Problem: A Call for Action. *Information* **2020**, *11*, 509. [CrossRef]
15. Casino, F.; Dasaklis, T.K.; Patsakis, C. A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telemat. Inform.* **2019**, *36*, 55–81. [CrossRef]
16. Yli-Huumo, J.; Ko, D.; Choi, S.; Park, S.; Smolander, K. Where Is Current Research on Blockchain Technology?—A Systematic Review. *PLoS ONE* **2016**, *11*, e0163477. [CrossRef]
17. Kaushik, A.; Choudhary, A.; Ektare, C.; Thomas, D.; Akram, S. Blockchain—Literature survey. In Proceedings of the RTE-ICT 2017—2nd IEEE International Conference on Recent Trends in Electronics, Information and Communication Technology, Proceedings, Bangalore, India, 19–20 May 2017; pp. 2145–2148.
18. Tranfield, D.; Denyer, D.; Smart, P. Towards a Methodology for Developing Evidence-Informed Management Knowledge by Means of Systematic Review. *Br. J. Manag.* **2003**, *14*, 207–222. [CrossRef]
19. Donthu, N.; Kumar, S.; Mukherjee, D.; Pandey, N.; Lim, W.M. How to conduct a bibliometric analysis: An overview and guidelines. *J. Bus. Res.* **2021**, *133*, 285–296. [CrossRef]

20. Chaudhary, A.S.; Mandalia, S.H.; Chaudhari, S.P.; Parmar, A.B. Lincoln Bibliometric Study of SAARC Countries Research Trends in Public Health Using Scopus Database Bibliometric Study of SAARC Countries Research Trends in Public Health Using. Available online: https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=10512&context=libphilprac (accessed on 4 June 2022).
21. Dulla, N.; Mishra, S.; Swain, S.C. *Global Exploration on Bibliometric Research Articles: A Bibliometric Analysis*; Sugyanta Priyadarshini's Lab: Bhubaneshwar, India, 2021; pp. 1–26.
22. Aziz, M.R.A.; Jali, M.Z.; Noor, M.N.M.; Sulaiman, S.; Harun, M.S.; Mustafar, M.Z.I. Bibliometric Analysis Of Literatures On Digital Banking And Financial Inclusion Between 2014–2020. *Libr. Philos. Pract.* **2021**, *2021*, 1–31.
23. Cobo, M.J.; Martínez, M.A.; Gutiérrez-Salcedo, M.; Fujita, H.; Herrera-Viedma, E. 25 years at Knowledge-Based Systems: A bibliometric analysis. *Knowl.-Based Syst.* **2015**, *80*, 3–13. [CrossRef]
24. Buttice, V.; Ughetto, E. What, Where, Who, and How? A Bibliometric Study of Crowdfunding Research. *IEEE Trans. Eng. Manag.* **2021**, 1–22. [CrossRef]
25. Sharma, T.K. Top 10 Real-World Applications of Blockchain Technology. Available online: https://www.blockchain-council.org/blockchain/top-10-real-world-applications-of-blockchain-technology/ (accessed on 2 April 2020).
26. Antonopoulos, A.M. *The Internet of Money: A Collection of Talks by Andreas Antonopoulos*, 1st ed.; CreateSpace Independent Publishing Platform: Scotts Valley, CA, USA, 2016; ISBN 978-1537000459.
27. Antonopoulos, A.M. *Mastering Bitcoin: Programming the Open Blockchain*, 2nd ed.; O'Reilly: Tokyo, Japan, 2017.
28. Jeffries, A. "Blockchain" is Meaningless. Available online: https://www.theverge.com/2018/3/7/17091766/blockchain-bitcoin-ethereum-cryptocurrency-meaning (accessed on 19 April 2020).
29. Damjan, M. The interface between blockchain and the real world. *Ragion Prat.* **2018**, *2018*, 379–406. [CrossRef]
30. Frankenreiter, J. The Limits of Smart Contracts. *J. Inst. Theor. Econ. JITE* **2019**, *175*, 149–162. [CrossRef]
31. Liu, X.; Chen, R.; Chen, Y.-W.; Yuan, S.-M. Off-chain Data Fetching Architecture for Ethereum Smart Contract. In Proceedings of the International Conference on Cloud Computing, Big Data and Blockchain, ICCBB 2018, Fuzhou, China, 15–17 November 2018.
32. Mammadzada, K.; Iqbal, M.; Milani, F.; García-Bañuelos, L.; Matulevičius, R. Blockchain Oracles: A Framework for Blockchain-Based Applications. Available online: https://zenodo.org/record/3605157#.Yp67suxBxPY (accessed on 4 June 2022).
33. Beniiche, A. A study of blockchain oracles. *arXiv* **2020**, arXiv:2004.07140.
34. Collins, P. What Is a Blockchain Oracle? Available online: https://medium.com/better-programming/what-is-a-blockchain-oracle-f5ccab8dbd72 (accessed on 21 October 2020).
35. Thevenard, J. Decentralised Oracles: A Comprehensive Overview. Available online: https://medium.com/fabric-ventures/decentralised-oracles-a-comprehensive-overview-d3168b9a8841 (accessed on 19 April 2020).
36. Huilgolkar, H. Razor Network: A Decentralized Oracle Platform. Available online: https://razor.network/whitepaper.pdf (accessed on 18 February 2021).
37. Breidenbach, L.; Cachin, C.; Chan, B.; Coventry, A.; Ellis, S.; Juels, A.; Koushanfar, F.; Miller, A.; Magauran, B.; Moroz, D.; et al. Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks; 2021. Available online: https://berkeley-defi.github.io/assets/material/Chainlink%202.0.pdf (accessed on 4 June 2022).
38. Caldarelli, G.; Ellul, J. Trusted academic transcripts on the blockchain: A systematic literature review. *Appl. Sci.* **2021**, *11*, 1842. [CrossRef]
39. Sharma, T.K. Centralized Oracles vs. Decentralized Oracles. Available online: https://www.blockchain-council.org/blockchain/centralized-oracles-vs-decentralized-oracles/ (accessed on 11 February 2021).
40. Larsens, A. Multisignature Wallet: How Does It Work? Available online: https://www.techzone360.com/topics/techzone/articles/2021/07/21/449512-multisignature-wallet-how-does-it-work.htm (accessed on 10 August 2021).
41. Dalovindj, U. The Oracle Problem. Available online: https://www.reddit.com/r/Bitcoin/comments/2p78kd/the_oracle_problem/ (accessed on 2 March 2020).
42. Kumar, A.; Liu, R.; Shan, Z. Is Blockchain a Silver Bullet for Supply Chain Management? Technical Challenges and Research Opportunities. *Decis. Sci.* **2020**, *51*, 8–37. [CrossRef]
43. Anadiotis, G. Off-Chain Reporting: Toward a New General Purpose Secure Compute Framework by Chainlink. Available online: https://www.zdnet.com/article/off-chain-reporting-towards-a-new-general-purpose-secure-compute-framework-by-chainlink/ (accessed on 19 March 2021).
44. Antonopoulos, A.M. The Killer App: Bananas on the Blockchain? Available online: https://aantonop.com/the-killer-app-bananas-on-theblockchain (accessed on 3 March 2020).
45. Sztorc, P. The Oracle Problem. Available online: https://www.infoq.com/presentations/blockchain-oracle-problems (accessed on 3 March 2020).
46. Caldarelli, G.; Ellul, J. The Blockchain Oracle Problem in Decentralized Finance—A Multivocal Approach. *Appl. Sci.* **2021**, *11*, 7572. [CrossRef]
47. Meinert, E.; Alturkistani, A.; Foley, K.A.; Osama, T.; Car, J.; Majeed, A.; Van Velthoven, M.; Wells, G.; Brindley, D. Blockchain Implementation in Health Care: Protocol for a Systematic Review. *JMIR Res. Protoc.* **2019**, *8*, e10994. [CrossRef]
48. Eskandari, S.; Salehi, M.; Gu, W.C.; Clark, J. SoK: Oracles from the Ground Truth to Market Manipulation. *arXiv* **2021**, arXiv:2106.00667.

49. Pahulje, M. What are NFTs & What Does It Mean for Supply Chains? Available online: https://blog.flexis.com/what-are-nfts-what-does-it-mean-for-supply-chains (accessed on 2 November 2021).
50. Kameir, C. NFT's Real World Use Case: Supply Chain | Hacker Noon. Available online: https://hackernoon.com/nfts-real-world-use-cases-exhibit-that-the-show-is-just-getting-started-mm3f33b8 (accessed on 2 November 2021).
51. Caldarelli, G. Wrapping Trust for Interoperability: A Preliminary Study of Wrapped Tokens. *Information* **2021**, *13*, 6. [CrossRef]
52. Finck, M.; Moscon, V. Copyright Law on Blockchains: Between New Forms of Rights Administration and Digital Rights Management 2.0. *IIC Int. Rev. Intellect. Prop. Compet. Law* **2019**, *50*, 77–108. [CrossRef]
53. Mik, E. Smart contracts: Terminology, technical limitations and real world complexity. *Law Innov. Technol.* **2017**, *9*, 269–300. [CrossRef]
54. Low, K.F.K.K.; Mik, E. Pause the Blockchain Legal Revolution. *Int. Comp. Law Q.* **2020**, *69*, 135–175. [CrossRef]
55. Haber, S.; Stornetta, S. How to timestamp a digital document—Original blockchain paper 1991. *J. Cryptol.* **1991**, *3*, 99–111. [CrossRef]
56. Martínez-Climent, C.; Zorio-Grima, A.; Ribeiro-Soriano, D. Financial return crowdfunding: Literature review and bibliometric analysis. *Int. Entrep. Manag. J.* **2018**, *14*, 527–553. [CrossRef]
57. Caviggioli, F.; Ughetto, E. A bibliometric analysis of the research dealing with the impact of additive manufacturing on industry, business and society. *Int. J. Prod. Econ.* **2019**, *208*, 254–268. [CrossRef]
58. Tenca, F.; Croce, A.; Ughetto, E. Business Angels Research in Entrepreneurial Finance: A Literature Review and a Research Agenda. *J. Econ. Surv.* **2018**, *32*, 1384–1413. [CrossRef]
59. Ramona, O.; Cristina, M.S.; Raluca, S. Bitcoin in the scientific literature—A bibliometric study. *Stud. Bus. Econ.* **2020**, *14*, 160–174. [CrossRef]
60. Xu, X.; Pautasso, C.; Zhu, L.; Gramoli, V.; Ponomarev, A.; Tran, A.B.; Chen, S. The blockchain as a software connector. In Proceedings of the 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), Venice, Italy, 5–8 April 2016; pp. 182–191. [CrossRef]
61. Dubey, A.; Hill, G.D.; Sirer, E.G.; Escriva, R. Weaver: A high-performance, transactional graph database based on refinable timestamps. *Proc. VLDB Endow.* **2016**, *9*, 852–863. [CrossRef]
62. Firdaus, A.; Razak, M.F.A.; Feizollah, A.; Hashem, I.A.T.; Hazim, M.; Anuar, N.B. The rise of "blockchain": Bibliometric analysis of blockchain study. *Scientometrics* **2019**, *120*, 1289–1331. [CrossRef]
63. Musigmann, B.; von der Gracht, H.; Hartmann, E. Blockchain Technology in Logistics and Supply Chain Management—A Bibliometric Literature Review From 2016 to January 2020. *IEEE Trans. Eng. Manag.* **2020**, *67*, 988–1007. [CrossRef]
64. Ante, L.; Steinmetz, F.; Fiedler, I. Blockchain and energy: A bibliometric analysis and review. *Renew. Sustain. Energy Rev.* **2021**, *137*, 110597. [CrossRef]
65. Ante, L. Smart contracts on the blockchain—A bibliometric analysis and review. *Telemat. Inform.* **2021**, *57*, 101519. [CrossRef]
66. Kochovski, P.; Gec, S.; Stankovski, V.; Bajec, M.; Drobintsev, P.D. Trust management in a blockchain based fog computing platform with trustless smart oracles. *Futur. Gener. Comput. Syst.* **2019**, *101*, 747–759. [CrossRef]
67. Pasdar, A.; Dong, Z.; Lee, Y.C. Blockchain Oracle Design Patterns. *arXiv* **2021**, arXiv:2106.09349.
68. Mühlberger, R.; Bachhofner, S.; Ferrer, E.C.; Di Ciccio, C.; Weber, I.; Wöhrer, M.; Zdun, U. Foundational Oracle Patterns: Connecting Blockchain to the Off-Chain World. In *Business Process Management: Blockchain and Robotic Process Automation Forum. BPM 2020. Lecture Notes in Business Information Processing*; Springer: Cham, Switzerland, 2020; Volume 393, pp. 35–51. [CrossRef]
69. Xu, X.; Pautasso, C.; Zhu, L.; Lu, Q.; Weber, I. A pattern collection for blockchain-based applications. In Proceedings of the EuroPLoP '18: Proceedings of the 23rd European Conference on Pattern Languages of Programs, Irsee, Germany, 4–8 July 2018. [CrossRef]
70. Xu, X.; Weber, I.; Staples, M. *Architecture for Blockchain Applications*; Springer: Berlin/Heidelberg, Germany, 2019; ISBN 9783030030346.
71. Xu, X.; Dilum Bandara, H.M.N.; Lu, Q.; Weber, I.; Bass, L.; Zhu, L. A Decision Model for Choosing Patterns in Blockchain-Based Applications. In Proceedings of the 2021 IEEE 18th International Conference on Software Architecture (ICSA), Stuttgart, Germany, 22–26 March 2021; pp. 47–57. [CrossRef]
72. Hendrix, C.; Lewis, R. Survey on Blockchain Privacy Challenges. Available online: https://www.semanticscholar.org/paper/Survey-on-Blockchain-Privacy-Challenges-Hendrix-Lewis/d2e0112490911091de39ea82fb97cd4f7aa829f4 (accessed on 18 March 2022).
73. Lin, S.Y.; Zhang, L.; Li, J.; Ji, L.L.; Sun, Y. A survey of application research based on blockchain smart contract. *Wirel. Netw.* **2022**, *28*, 635–690. [CrossRef]
74. Zhang, W.; Wei, L.; Li, S.; Liu, Y.; Cheung, S.C. Đarcher: Detecting on-chain-off-chain synchronization bugs in decentralized applications. In Proceedings of the ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, 23–28 August 2021; pp. 553–565. [CrossRef]
75. Turksonmez, K.; Furtak, M.; Wittie, M.P.; Millman, D.L. Two Ways Gas Price Oracles Miss the Mark. In Proceedings of the 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), Barcelona, Spain, 23–25 August 2021. [CrossRef]
76. Mars, R.; Abid, A.; Cheikhrouhou, S.; Kallel, S. A machine learning approach for gas price prediction in ethereum blockchain. In Proceedings of the Proceedings—2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC 2021, Madrid, Spain, 12–16 July 2021; pp. 156–165.

77. Pierro, G.A.; Rocha, H.; Ducasse, S.; Marchesi, M.; Tonelli, R. A user-oriented model for Oracles' Gas price prediction. *Futur. Gener. Comput. Syst.* **2022**, *128*, 142–157. [CrossRef]
78. Mezquita, Y.; Valdeolmillos, D.; González-Briones, A.; Prieto, J.; Corchado, J.M. Legal aspects and emerging risks in the use of smart contracts based on blockchain. *Commun. Comput. Inf. Sci.* **2019**, *1027*, 525–535. [CrossRef]
79. Sheldon, M.D. Preparing Auditors for the Blockchain Oracle Problem. *Curr. Issues Audit.* **2021**, *15*, P27–P39. [CrossRef]
80. Murimi, R.M.; Wang, G.G. On elastic incentives for blockchain oracles. *J. Database Manag.* **2021**, *32*, 1–26. [CrossRef]
81. Yutaka, K.; Fujihara, A. ken-system: Experimental performance evaluation on avoidance of blockchain oracle problem using collective intelligence. In *IEICE Technology Report*; No. 414, IN2020-74; The Institute of Electronics, Information and Communication Engineers: Tokyo, Japan, 2021; Volume 120, pp. 120–125.
82. Kaleem, M.; Shi, W. Demystifying Pythia: A Survey of ChainLink Oracles Usage on Ethereum. *arXiv* **2021**, arXiv:2101.06781.
83. Kumar, M.; Nikhil, N.; Singh, R. Decentralising Finance using Decentralised Blockchain Oracles. In Proceedings of the 2020 International Conference for Emerging Technology (INCET), Belgaum, India, 5–7 June 2020; pp. 1–4.
84. Tjiam, K.; Wang, R.; Chen, H.; Liang, K. Your Smart Contracts Are Not Secure: Investigating Arbitrageurs and Oracle Manipulators in Ethereum. In Proceedings of the CYSARM '21: Proceedings of the 3rd Workshop on Cyber-Security Arms Race, Online, 15 November 2021; pp. 25–35. [CrossRef]
85. Wang, B.; Liu, H.; Liu, C.; Yang, Z.; Ren, Q.; Zheng, H.; Lei, H. BLOCKEYE: Hunting For DeFi Attacks on Blockchain. *arXiv* **2021**, arXiv:2103.02873.
86. Bai, L. Oracle's Application in Finance. In *Big Data Analytics for Cyber-Physical System in Smart City*; Springer: Singapore, 2021.
87. Omar, I.A.; Hasan, H.R.; Jayaraman, R.; Salah, K.; Omar, M. Implementing decentralized auctions using blockchain smart contracts. *Technol. Forecast. Soc. Chang.* **2021**, *168*, 120786. [CrossRef]
88. Lafourcade, P.; Nopere, M.; Picot, J.; Pizzuti, D.; Lafourcade, P.; Nopere, M.; Picot, J.; Pizzuti, D.; Roudeix, E.; Analysis, S. *Security Analysis of Auctionity: A Blockchain Based E-Auction*; Springer: Berlin/Heidelberg, Germany, 2019.
89. Belchior, R.; Vasconcelos, A.; Guerreiro, S.; Correia, M. A Survey on Blockchain Interoperability: Past, Present, and Future Trends. *ACM Comput. Surv.* **2021**, *54*, 1–41. [CrossRef]
90. Hei, Y.; Li, D.; Zhang, C.; Liu, J.; Liu, Y.; Wu, Q. Practical AgentChain: A compatible cross-chain exchange system. *Futur. Gener. Comput. Syst.* **2022**, *130*, 207–218. [CrossRef]
91. Di Ciccio, C.; Meroni, G.; Plebani, P. On the adoption of blockchain for business process monitoring. *Softw. Syst. Model.* **2022**, *21*, 1–23. [CrossRef]
92. Ladleif, J.; Weske, M. Time in Blockchain-Based Process Execution. In Proceedings of the 24th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2020, Eindhoven, The Netherlands, 5–8 October 2020; pp. 217–226. [CrossRef]
93. Ladleif, J.; Weber, I.; Weske, M. *External Data Monitoring Using Oracles in Blockchain-Based Process Execution*; Springer: Berlin/Heidelberg, Germany, 2020; Volume 393, ISBN 9783030587789.
94. Ladleif, J.; Weske, M. Which Event Happened First? Deferred Choice on Blockchain Using Oracles. *arXiv* **2021**, arXiv:2104.10520. [CrossRef]
95. Carminati, B.; Ferrari, E.; Rondanini, C. Blockchain as a platform for secure inter-organizational business processes. In Proceedings of the 2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC), Philadelphia, PA, USA, 18–20 October 2018; pp. 122–129. [CrossRef]
96. Carminati, B.; Rondanini, C.; Ferrari, E. Confidential Business Process Execution on Blockchain. In Proceedings of the Proceedings—2018 IEEE International Conference on Web Services, ICWS 2018—Part of the 2018 IEEE World Congress on Services, San Francisco, CA, USA, 2–7 July 2018; pp. 58–65.
97. Lu, W.; Li, X.; Xue, F.; Zhao, R.; Wu, L.; Yeh, A.G.O. Exploring smart construction objects as blockchain oracles in construction supply chain management. *Autom. Constr.* **2021**, *129*, 103816. [CrossRef]
98. Victor, F.; Zickau, S. Geofences on the blockchain: Enabling decentralized location-based services. In Proceedings of the IEEE International Conference on Data Mining Workshops, ICDMW, Singapore, 17–20 November 2018; pp. 97–104.
99. Moudoud, H.; Cherkaoui, S.; Khoukhi, L. An IoT Blockchain Architecture Using Oracles and Smart Contracts: The Use-Case of a Food Supply Chain. In Proceedings of the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC, Istanbul, Turkey, 8–11 November 2019.
100. Sánchez-Gómez, N.; Risoto, M.M.; Ramos-Cueli, J.M.; Wojdyński, T.; Lizcano, D.; Torres-Valderrama, J. The current limitations of blockchain traceability: Challenges from industry. In Proceedings of the 16th International Conference on Web Information Systems and Technologies, Budapest, Hungary, 3–5 November 2020; pp. 373–380. [CrossRef]
101. Marbouh, D.; Abbasi, T.; Maasmi, F.; Omar, I.A.; Debe, M.S.; Salah, K.; Jayaraman, R.; Ellahham, S. Blockchain for COVID-19: Review, Opportunities, and a Trusted Tracking System. *Arab. J. Sci. Eng.* **2020**, *45*, 9895–9911. [CrossRef]
102. Powell, W.; Foth, M.; Cao, S.; Natanelov, V. Garbage in garbage out: The precarious link between IoT and blockchain in food supply chains. *J. Ind. Inf. Integr.* **2022**, *25*, 100261. [CrossRef]
103. Caldarelli, G.; Zardini, A.; Rossignoli, C. Blockchain adoption in the fashion sustainable supply chain: Pragmatically addressing barriers. *J. Organ. Chang. Manag.* **2021**, *34*, 507–524. [CrossRef]
104. Madine, M.M.; Battah, A.A.; Yaqoob, I.; Salah, K.; Jayaraman, R.; Al-Hammadi, Y.; Pesic, S.; Ellahham, S. Blockchain for Giving Patients Control over Their Medical Records. *IEEE Access* **2020**, *8*, 193102–193115. [CrossRef]

105. Madine, M.M.; Salah, K.; Jayaraman, R.; Yaqoob, I.; Al-Hammadi, Y.; Ellahham, S.; Calyam, P. Fully Decentralized Multi-Party Consent Management for Secure Sharing of Patient Health Records. *IEEE Access* **2020**, *8*, 225777–225791. [CrossRef]
106. Gonçalves, J.P.B.; de Resende, H.C.; Villaca, R.S.; Municio, E.; Both, C.B.; Marquez-Barja, J.M. Distributed Network Slicing Management Using Blockchains in E-Health Environments. *Mob. Netw. Appl.* **2021**, *26*, 2111–2122. [CrossRef]
107. Beniiche, A.; Rostami, S.; Maier, M. Robonomics in the 6G Era: Playing the Trust Game with On-Chaining Oracles and Persuasive Robots. *IEEE Access* **2021**, *9*, 46949–46959. [CrossRef]
108. Sata, B.; Berlanga, A.; Chanel, C.P.C.; Lacan, J. Connecting AI-based Oracles to Blockchains via an Auditable Auction Protocol. In Proceedings of the 2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), Paris, France, 27–30 September 2021; pp. 23–24. [CrossRef]
109. El Fezzazi, A.; Adadi, A.; Berrada, M. Towards a Blockchain based Intelligent and Secure Voting. In Proceedings of the 2021 Fifth International Conference on Intelligent Computing in Data Sciences (ICDS), Fez, Morocco, 20–22 October 2021. [CrossRef]
110. Richard; Surya, M.M.; Wibowo, A.C. Converging artificial intelligence and blockchain technology using oracle contract in ethereum blockchain platform. In Proceedings of the 2020 5th International Conference on Informatics and Computing, ICIC 2020, Gorontalo, Indonesia, 3–4 November 2020.
111. Gordon, G. Provenance and Authentication of Oracle Sensor Data with Block Chain Lightweight Wireless Network Authentication Scheme for Constrained Oracle Sensors. Master's Thesis, Saint Mary's University, Halifax, NS, Canada, January 2017.
112. Vári-Kakas, S.; Poszet, O.; Mirela Pater, A.; Valentina Moisi, E.; Vári-Kakas, A. Issues Related to the Use of Blockchains in IoT Applications. In Proceedings of the 2021 16th International Conference on Engineering of Modern Electric Systems (EMES), Oradea, Romania, 10–11 June 2021. [CrossRef]
113. Shi, P.; Wang, H.; Yang, S.; Chen, C.; Yang, W. Blockchain-based trusted data sharing among trusted stakeholders in IoT. *Softw.-Pract. Exp.* **2019**, *51*, 2051–2064. [CrossRef]
114. Rondanini, C.; Carminati, B.; Ferrari, E. Confidential discovery of IoT devices through blockchain. In Proceedings of the Proceedings—2019 IEEE International Congress on Internet of Things, ICIOT 2019—Part of the 2019 IEEE World Congress on Services, Milan, Italy, 8–13 July 2019; pp. 1–8.
115. Moudoud, H.; Cherkaoui, S.; Khoukhi, L. Towards a Scalable and Trustworthy Blockchain: IoT Use Case. In Proceedings of the ICC 2021—IEEE International Conference on Communications, Montreal, QC, Canada, 14–23 June 2021; pp. 1–6. [CrossRef]
116. Kochovski, P.; Stankovski, V.; Gec, S.; Faticanti, F.; Savi, M.; Siracusa, D.; Kum, S. Smart Contracts for Service-Level Agreements in Edge-to-Cloud Computing. *J. Grid Comput.* **2020**, *18*, 673–690. [CrossRef]
117. Tao, X.; Hafid, A.S. ChainSensing: A Novel Mobile Crowdsensing Framework with Blockchain. *IEEE Internet Things J.* **2022**, *9*, 2999–3010. [CrossRef]
118. Taghavi, M.; Bentahar, J.; Otrok, H.; Bakhtiyari, K. A Blockchain-Based Model for Cloud Service Quality Monitoring. *IEEE Trans. Serv. Comput.* **2020**, *13*, 276–288. [CrossRef]
119. Comuzzi, M.; Cappiello, C.; Meroni, G. An Empirical Evaluation of Smart Contract-Based Data Quality Assessment in Ethereum. *Lect. Notes Bus. Inf. Process.* **2021**, *428*, 51–66. [CrossRef]
120. Battah, A.A.; Madine, M.M.; Alzaabi, H.; Yaqoob, I.; Salah, K.; Jayaraman, R. Blockchain-based multi-party authorization for accessing iPFS encrypted data. *IEEE Access* **2020**, *8*, 196813–196825. [CrossRef]
121. Battah, A.; Iraqi, Y.; Damiani, E. Blockchain-based reputation systems: Implementation challenges and mitigation. *Electronics* **2021**, *10*, 289. [CrossRef]
122. Mitra, D.; Tauz, L.; Dolecek, L. *Communication-Efficient LDPC Code Design for Data Availability Oracle in Side Blockchains*; IEEE: Piscataway, NJ, USA, 2021.
123. Gao, Z.; Li, H.; Xiao, K.; Wang, Q. Cross-chain oracle based data migration mechanism in heterogeneous blockchains. In Proceedings of the Proceedings—International Conference on Distributed Computing Systems, Singapore, 29 November–1 December 2020; pp. 1263–1268.
124. Ouyang, R.; Ouyang, R. HBRO: A Registration Oracle Scheme for Digital Rights Management Based on Heterogeneous Blockchains. *Commun. Netw.* **2021**, *14*, 45–67. [CrossRef]
125. Al Zaabi, A.; Yeun, C.Y.; Damiani, E. Trusting Testcases Using Blockchain-Based Repository Approach. *Symmetry* **2021**, *13*, 2024. [CrossRef]
126. Yeh, L.Y.; Shen, N.X.; Hwang, R.H. Blockchain-Based Privacy-Preserving and Sustainable Data Query Service Over 5G-VANETs. *IEEE Trans. Intell. Transp. Syst.* **2022**, 1–13. [CrossRef]
127. Lv, P.; Zhang, X.; Liu, J.; Wei, T.; Xu, J. Blockchain Oracle-Based Privacy Preservation and Reliable Identification for Vehicles. In Proceedings of the International Conference on Wireless Algorithms, Systems, and Applications, Nanjing, China, 25–27 June 2021; pp. 512–520.
128. Osterland, T.; Rose, T. Oracle-based process automation in DLT dominated ecosystems with an application to German waterway transportation. In Proceedings of the 2nd Asia Service Sciences and Software Engineering Conference, Macau, China, 24–26 February 2021; pp. 130–136.
129. Antal, C.; Cioara, T.; Antal, M.; Mihailescu, V.; Mitrea, D.; Anghel, I.; Salomie, I.; Raveduto, G.; Bertoncini, M.; Croce, V.; et al. Blockchain based decentralized local energy flexibility market. *Energy Rep.* **2021**, *7*, 5269–5288. [CrossRef]
130. Zeiselmair, A.; Steinkopf, B.; Gallersdörfer, U.; Bogensperger, A.; Matthes, F. Analysis and Application of Verifiable Computation Techniques in Blockchain Systems for the Energy Sector. *Front. Blockchain* **2021**, *37*, 725322. [CrossRef]

131. Weixian, W.; Ping, C.; Mingyu, P.; Xianglong, L.; Zhuoqun, L.; Ruixin, H. Design of Collaborative Control Scheme between On-chain and Off-chain Power Data. In Proceedings of the 2021 IEEE 4th International Conference on Information Systems and Computer Aided Education (ICISCAE), Dalian, China, 24–26 September 2021; pp. 1–6.
132. Park, J.; Kim, H.; Kim, G.; Ryou, J. Smart contract data feed framework for privacy-preserving oracle system on blockchain. *Computers* **2021**, *10*, 7. [CrossRef]
133. Chen, L.; Yuan, R.; Xia, Y. Tora: A Trusted Blockchain Oracle Based on a Decentralized TEE Network. In Proceedings of the 2021 IEEE International Conference on Joint Cloud Computing (JCC), Oxford, UK, 23–26 August 2021; pp. 28–33. [CrossRef]
134. Song, J. The Truth about Smart Contracts. Available online: https://medium.com/@jimmysong/the-truth-about-smart-contracts-ae825271811f (accessed on 2 March 2020).
135. Wang, H.; Guo, C.; Cheng, S. LoC—A new financial loan management system based on smart contracts. *Futur. Gener. Comput. Syst.* **2019**, *100*, 648–655. [CrossRef]
136. Tucci-Piergiovanni, S. Invited Paper: On the Characterization of Blockchain Consensus Under Incentives. In *International Symposium on Stabilizing, Safety, and Security of Distributed Systems*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 1–15.
137. Mhaisen, N.; Fetais, N.; Erbad, A.; Mohamed, A.; Guizani, M. To chain or not to chain: A reinforcement learning approach for blockchain-enabled IoT monitoring applications. *Futur. Gener. Comput. Syst.* **2020**, *111*, 39–51. [CrossRef]
138. Tsai, W.-T.; Xiang, W.; Wang, R.; Deng, E. *LSO: A Dynamic and Scalable Blockchain Structuring Framework*; Springer: Berlin/Heidelberg, Germany, 2021; Volume 1385.
139. Roussille, H.; Gürcan, Ö.; Michel, F. AGR4BS: A Generic Multi-Agent Organizational Model for Blockchain Systems. *Big Data Cogn. Comput.* **2021**, *6*, 1. [CrossRef]
140. Grooteman, B.; Eindhoven, T.U. Providing Trusted Datafeeds to the Blockchain; 2019. Available online: https://pure.tue.nl/ws/portalfiles/portal/145685273/Thesis_Bram_Grooteman.pdf (accessed on 4 June 2022).
141. Ritzdorf, H.; Wust, K.; Gervais, A.; Felley, G.; Capkun, S. TLS-N: Non-repudiation over TLS Enabling Ubiquitous Content Signing. In Proceedings of the Network and Distributed Systems Security (NDSS) Symposium 2018, San Diego, CA, USA, 18–21 February 2018. [CrossRef]
142. Liu, X.; Muhammad, K.; Lloret, J.; Chen, Y.-W.; Yuan, S.-M. Elastic and cost-effective data carrier architecture for smart contract in blockchain. *Futur. Gener. Comput. Syst.* **2019**, *100*, 590–599. [CrossRef]
143. Heiss, J.; Eberhardt, J.; Tai, S. From oracles to trustworthy data on-chaining systems. In Proceedings of the Proceedings—2019 2nd IEEE International Conference on Blockchain, Blockchain 2019, Atlanta, GA, USA, 14–17 July 2019; pp. 496–503.
144. Abusalah, H.; Alwen, J.; Cohen, B.; Khilko, D.; Pietrzak, K.; Reyzin, L. Beyond Hellman's Time-Memory Trade-Offs with Applications to Proofs of Space. 2017. Available online: https://eprint.iacr.org/2017/893.pdf (accessed on 4 June 2022).
145. Liu, X.; Feng, J. Trusted Blockchain Oracle Scheme Based on Aggregate Signature. *J. Comput. Commun.* **2021**, *09*, 95–109. [CrossRef]
146. Wang, Y.; Liu, H.; Wang, J.; Wang, S. *Efficient Data Interaction of Blockchain Smart Contract with Oracle Mechanism*; IEEE: Piscataway, NJ, USA, 2020; pp. 1000–1003.
147. Patel, N.S.; Bhattacharya, P.; Patel, S.B.; Tanwar, S.; Kumar, N.; Song, H. Blockchain-envisioned trusted random oracles for IoT-enabled Probabilistic Smart Contracts. *IEEE Internet Things J.* **2021**, *8*, 14797–14809. [CrossRef]
148. Poblet, M.; Allen, D.W.E.; Konashevych, O.; Lane, A.M.; Diaz Valdivia, C.A. From Athens to the Blockchain: Oracles for Digital Democracy. *Front. Blockchain* **2020**, *3*, 575662. [CrossRef]
149. Adams, B.; Tomko, M. A Critical Look at Cryptogovernance of the Real World: Challenges for Spatial Representation and Uncertainty on the Blockchain. Available online: https://drops.dagstuhl.de/opus/volltexte/2018/9346/ (accessed on 4 June 2022).
150. Merlini, M.; Veira, N.; Berryhill, R.; Veneris, A. On Public Decentralized Ledger Oracles via a Paired-Question Protocol. In Proceedings of the ICBC 2019—IEEE International Conference on Blockchain and Cryptocurrency, Seoul, Korea, 14–17 May 2019; pp. 337–344.
151. Lo, S.K.; Xu, X.; Staples, M.; Yao, L. Reliability analysis for blockchain oracles. *Comput. Electr. Eng.* **2020**, *83*, 106582. [CrossRef]
152. Basile, D.; Goretti, V.; Di Ciccio, C.; Kirrane, S. Enhancing Blockchain-Based Processes with Decentralized Oracles. In *Business Process Management: Blockchain and Robotic Process Automation Forum. BPM 2021. Lecture Notes in Business Information Processing*; González Enríquez, J., Debois, S., Fettke, P., Plebani, P., van de Weerd, I., Weber, I., Eds.; Springer: Cham, Switzerland, 2021; Volume 1, pp. 102–118. [CrossRef]
153. Goyal, V.; Raizes, J.; Soni, P. Time-Traveling Simulators Using Blockchains and Their Applications. Available online: https://eprint.iacr.org/2022/035 (accessed on 4 June 2022).
154. Ellul, J.; Pace, G.J. Towards External Calls for Blockchain and Distributed Ledger Technology. 2021. Available online: https://www.researchgate.net/publication/351497108_Towards_External_Calls_for_Blockchain_and_Distributed_Ledger_Technology (accessed on 4 June 2022).
155. Bruschi, F.; Rana, V.; Pagani, A.; Sciuto, D. Tunneling Trust into the Blockchain: A Merkle Based Proof System for Structured Documents. *IEEE Access* **2021**, *9*, 103758–103771. [CrossRef]
156. Simunic, S.; Bernaca, D.; Lenac, K. Verifiable Computing Applications in Blockchain. *IEEE Access* **2021**, *9*, 156729–156745. [CrossRef]

157. Chatterjee, K.; Goharshady, A.K.; Pourdamghani, A. Probabilistic smart contracts: Secure randomness on the blockchain. In Proceedings of the ICBC 2019—IEEE International Conference on Blockchain and Cryptocurrency, Seoul, Korea, 14–17 May 2019; pp. 403–412.
158. Gallersdorfer, U.; Matthes, F. Towards Valid Use Cases: Requirements and Supporting Characteristics of Proper Blockchain Applications. In Proceedings of the 2020 7th International Conference on Software Defined Systems, SDS 2020, Paris, France, 20–23 April 2020; pp. 202–207.
159. Schaad, A.; Reski, T.; Winzenried, O. Integration of a Secure Physical Element as a Trusted Oracle in a Hyperledger Blockchain. In Proceedings of the 16th International Joint Conference on e-Business and Telecommunications, Prague, Czech Republic, 26–28 July 2019; SCITEPRESS—Science and Technology Publications: Prague, Czech Republic, 2019; pp. 498–503.
160. Truong, N.; Lee, G.M.; Sun, K.; Guitton, F.; Guo, Y.K. A blockchain-based trust system for decentralised applications: When trustless needs trust. *Futur. Gener. Comput. Syst.* **2021**, *124*, 68–79. [CrossRef]
161. Motaqy, Z.; Almashaqbeh, G.; Bahrak, B.; Yazdani, N. Bet and Attack: Incentive Compatible Collaborative Attacks Using Smart Contracts. In *Decision and Game Theory for Security. GameSec 2021. Lecture Notes in Computer Science*; Bošanský, B., Gonzalez, C., Rass, S., Sinha, A., Eds.; Springer: Cham, Switzerland, 2021; Volume 13061, pp. 293–313. [CrossRef]
162. Wang, S.; Lu, H.; Sun, X.; Yuan, Y.; Wang, F.-Y. A Novel Blockchain Oracle Implementation Scheme Based on Application Specific Knowledge Engines. In Proceedings of the 2019 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), Zhengzhou, China, 11–13 October 2019; IEEE: Piscataway, NJ, USA, 2019; Volume 2, pp. 258–262.
163. Wang, S.; Yu, X. Research on Trusted Identification of Blockchain Uploaded Data. In Proceedings of the 2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA), Dalian, China, 27–29 June 2020; pp. 1036–1044. [CrossRef]
164. Van Der Laan, B.; Ersoy, O.; Erkin, Z. Muscle: Authenticated external data retrieval from multiple sources for smart contracts. In Proceedings of the ACM Symposium on Applied Computing, New York, NY, USA, 8–12 April 2019; Volume Part F1477, pp. 382–391.
165. George, W.; Lesaege, C. A Smart Contract Oracle for Approximating Real-World, Real Number Values. Available online: https://drops.dagstuhl.de/opus/volltexte/2020/11970/pdf/OASIcs-Tokenomics-2019-6.pdf (accessed on 4 June 2022).
166. Ma, L.; Kaneko, K.; Sharma, S.; Sakurai, K. Reliable decentralized oracle with mechanisms for verification and disputation. In Proceedings of the 2019 7th International Symposium on Computing and Networking Workshops, CANDARW 2019, Nagasaki, Japan, 26–29 November 2019; pp. 346–352.
167. Chen, Y.-J.; Wu, J.-L.; Hsieh, Y.-C.; Hsueh, C.-W. An oracle-based on-chain privacy. *Computers* **2020**, *9*, 69. [CrossRef]
168. Woo, S.; Song, J.; Park, S. A distributed oracle using intel SGX for blockchain-based IoT applications. *Sensors* **2020**, *20*, 2725. [CrossRef]
169. Goel, N.; van Schreven, C.; Filos-Ratsikas, A.; Faltings, B. Infochain: A Decentralized, Trustless and Transparent Oracle on Blockchain. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, Yokohama, Japan, 11–17 July 2020; pp. 4604–4610.
170. Muller, M.; Rodriguez Garzon, S.; Kupper, A. COST: A Consensus-Based Oracle Protocol for the Secure Trade of Digital Goods. In Proceedings of the 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS), Oxford, UK, 3–6 August 2020; pp. 72–81. [CrossRef]
171. Sheng, P.; Xue, B.; Kannan, S.; Viswanath, P. ACeD: Scalable Data Availability Oracle. *arXiv* **2020**, arXiv:2011.00102.
172. Cai, Y.; Fragkos, G.; Tsiropoulou, E.E.; Veneris, A. A Truth-Inducing Sybil Resistant Decentralized Blockchain Oracle. In Proceedings of the 2020 2nd Conference on Blockchain Research and Applications for Innovative Networks and Services, BRAINS 2020, Paris, France, 28–30 September 2020; pp. 128–135.
173. Adler, J.; Berryhill, R.; Veneris, A.; Poulos, Z.; Veira, N.; Kastania, A. Astraea: A Decentralized Blockchain Oracle. In Proceedings of the 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 30 July–3 August 2018; pp. 1145–1152.
174. Nelaturu, K.; Adler, J.; Merlini, M.; Berryhill, R.; Veira, N.; Poulos, Z.; Veneris, A. On Public Crowdsource-Based Mechanisms for a Decentralized Blockchain Oracle. *IEEE Trans. Eng. Manag.* **2020**, *67*, 1444–1458. [CrossRef]
175. Cai, Y.; Irtija, N.; Tsiropoulou, E.E.; Veneris, A. *Truthful Decentralized Blockchain Oracles*; Wiley: Hoboken, NJ, USA, 2021. [CrossRef]
176. Sober, M.; Scaffino, G.; Spanring, C.; Schulte, S. A Voting-Based Blockchain Interoperability Oracle. *Blockchain* **2022**, 160–169. [CrossRef]
177. Chishti, M.S.; Sufyan, F.; Banerjee, A. Decentralized On-Chain Data Access via Smart Contracts in Ethereum Blockchain. *IEEE Trans. Netw. Serv. Manag.* **2021**, 174–187. [CrossRef]
178. Ducrée, J. Digital Twin: An Oracle for Efficient Crowdsourcing of Research & Technology Development through Blockchain. *Preprints* **2021**, 2021100148. [CrossRef]
179. Chen, S.; Zhu, J.; Lin, Z.; Huang, J.; Tang, Y. How to Make Smart Contract Smarter. In *CCF Conference on Computer Supported Cooperative Work and Social Computing*; Springer: Berlin/Heidelberg, Germany, 2021; Volume 1330, pp. 705–713. [CrossRef]
180. Mohsin, F.; Zhao, X.; Hong, Z.; de Mel, G.; Xia, L.; Seneviratne, O. Ontology Aided Smart Contract Execution for Unexpected Situations. Available online: http://ceur-ws.org/Vol-2599/paper1.pdf (accessed on 4 June 2022).

181. Berger, B.; Huber, S.; Pfeifhofer, S. OraclesLink: An architecture for secure oracle usage. In Proceedings of the 2020 2nd International Conference on Blockchain Computing and Applications, BCCA 2020, Antalya, Turkey, 2–5 November 2020; pp. 66–72.
182. Adamik, F.; Kosta, S. SmartExchange: Decentralised Trustless Cryptocurrency Exchange. In *Lecture Notes in Business Information Processing*; Springer: Berlin/Heidelberg, Germany, 2019; Volume 339, pp. 356–367.
183. Ji, Y.; Gu, W.; Chen, F.; Xiao, X.; Sun, J.; Liu, S.; He, J.; Li, Y.; Zhang, K.; Mei, F.; et al. SEBF: A single-chain based extension model of blockchain for fintech. In Proceedings of the IJCAI International Joint Conference on Artificial Intelligence; Yokohama, Japan, 11–17 July 2020; pp. 4497–4505.
184. Antonio Pierro, G.; Rocha, H.; Tonelli, R.; Ducasse, S. Are the Gas Prices Oracle Reliable? A Case Study using the EthGasStation. In Proceedings of the IWBOSE 2020, 2020 IEEE 3rd International Workshop on Blockchain Oriented Software Engineering, London, ON, Canada, 18 February 2020; pp. 1–8.
185. Galanis, S. No Trade in a Blockchain. *SSRN Electron. J.* **2020**, 1–13. [CrossRef]
186. Yu, L.; Zichichi, M.; Markovich, R.; Najjar, A. Enhancing Trust in Trust Services: Towards an Intelligent Human-input-based Blockchain Oracle (IHiBO). In Proceedings of the 55th Hawaii International Conference on System Sciences, HICSS 2022, Virtual Event, Maui, HI, USA, 4–7 January 2022. [CrossRef]
187. Nassar, M.; Salah, K.; ur Rehman, M.H.; Svetinovic, D. Blockchain for explainable and trustworthy artificial intelligence. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2020**, *10*, 1–13. [CrossRef]
188. Lopes, V.; Pereira, N.; Alexandre, L.A. Robot workspace monitoring using a blockchain-based 3D vision approach. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, Long Beach, CA, USA, 16–17 June 2019; pp. 2812–2820.
189. Yu, L.; Zichichi, Z.; Markovich, R.; Najjar, A. Argumentation in Trust Services within a Blockchain Environment. Available online: https://orbilu.uni.lu/bitstream/10993/50021/1/Argumentation%20in%20Trust%20Services%20within%20a%20Blockchain%20Environment.pdf (accessed on 4 June 2022).
190. Alowayed, Y.; Canini KAUST Pedro Marcos, M.; Chiesa, M.; Barcellos UFRGS, M.; Canini, M.; Marcos, P.; Barcellos, M. Picking a Partner: A Fair Blockchain Based Scoring Protocol for Autonomous Systems. In Proceedings of the ANRW'18, Montreal, QC, Canada, 16 July 2018. [CrossRef]
191. Autili, M.; Gallo, F.; Inverardi, P.; Pompilio, C.; Tivoli, M. Introducing trust in service-oriented distributed systems through blockchain. In Proceedings of the 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 27–30 October 2019; pp. 149–154. [CrossRef]
192. Adjei-Arthur, B.; Gao, J.; Xia, Q.; da Silva Tavares, E.; Xia, H.; Amofa, S.; Wang, Y. A blockchain-adaptive contractual approach for multi-contracting organizational entities. *Futur. Gener. Comput. Syst.* **2022**, *132*, 93–107. [CrossRef]
193. Hasan, M.; Ogan, K.; Starly, B. Hybrid Blockchain Architecture for Cloud Manufacturing-as-a-service (CMaaS) Platforms with Improved Data Storage and Transaction Efficiency. *Procedia Manuf.* **2021**, *53*, 594–605. [CrossRef]
194. Zhang, C.; Zhu, L.; Xu, C.; Sharif, K. PRVB: Achieving Privacy-Preserving and Reliable Vehicular Crowdsensing via Blockchain Oracle. *IEEE Trans. Veh. Technol.* **2021**, *70*, 831–843. [CrossRef]
195. Fujihara, A. *Proposing a Blockchain-Based Open Data Platform and Its Decentralized Oracle*; Springer International Publishing: Berlin/Heidelberg, Germany, 2020; Volume 1035, ISBN 9783030290344.
196. Drungilas, V.; Vaičiukynas, E.; Jurgelaitis, M.; Butkienė, R.; Čeponienė, L. Towards blockchain-based federated machine learning: Smart contract for model inference. *Appl. Sci.* **2021**, *11*, 1010. [CrossRef]
197. Chaer, A.; Salah, K.; Lima, C.; Ray, P.P.; Sheltami, T. Blockchain for 5G: Opportunities and challenges. In Proceedings of the 2019 IEEE Globecom Workshops, GC Wkshps 2019—Proceedings, Waikoloa, HI, USA, 9–13 December 2019.
198. Anceaume, E.; Del Pozzo, A.; Ludinard, R.; Potop-Butucaru, M.; Tucci-Piergiovanni, S. Blockchain abstract data type. In Proceedings of the SPAA '19: 31st ACM Symposium on Parallelism in Algorithms and Architectures, Phoenix, AZ, USA, 22–24 June 2019; pp. 349–358. [CrossRef]
199. Zhao, J.; Lin, Z.; Huang, X.; Zhang, Y.; Xiang, S. TrustCA: Achieving Certificate Transparency through Smart Contract in Blockchain Platforms. In Proceedings of the 2020 International Conference on High Performance Big Data and Intelligent Systems, HPBD and IS 2020, Shenzhen, China, 23 May 2020.
200. Paik, H.Y.; Xu, X.; Bandara, H.M.N.D.; Lee, S.U.; Lo, S.K. Analysis of data management in blockchain-based systems: From architecture to governance. *IEEE Access* **2019**, *7*, 186091–186107. [CrossRef]
201. Moudoud, H.; Cherkaoui, S.; Khoukhi, L. An Overview of Blockchain and 5G Networks. In *Computational Intelligence in Recent Communication Networks. EAI/Springer Innovations in Communication and Computing*; Ouaissa, M., Boulouard, Z., Ouaissa, M., Guermah, B., Eds.; Springer: Cham, Switzerland, 2022; pp. 1–20. [CrossRef]
202. Yeh, L.Y.; Lu, P.J.; Huang, S.H.; Huang, J.L. SOChain: A Privacy-Preserving DDoS Data Exchange Service over SOC Consortium Blockchain. *IEEE Trans. Eng. Manag.* **2020**, *67*, 1487–1500. [CrossRef]
203. Müller, M.; Garzon, S.R. Blockchain-Based Trusted Cross-Organizational Deliveries of Sensor-Equipped Parcels. In Proceedings of the Parallel Processing Workshops: Euro-Par 2019 International Workshops, Göttingen, Germany, 26–30 August 2019.
204. Maier, M. 6G as if People Mattered: From Industry 4.0 toward Society 5.0: (Aper). In Proceedings of the Proceedings—International Conference on Computer Communications and Networks, ICCCN, Athens, Greece, 19–22 July 2021.

205. Lin, Y.; Gao, Z.; Shi, W.; Wang, Q.; Li, H.; Wang, M.; Yang, Y.; Rui, L. A Novel Architecture Combining Oracle with Decentralized Learning for IIoT. *IEEE Internet Things J.* **2022**, 1. [CrossRef]
206. Raheman, F.; Bhagat, T.; Dayma, A.; Raheman, A.; Anwar, S. The Blockchain Paradox: Testing the DISC Hypothesis. Available online: https://ijisrt.com/assets/upload/files/IJISRT21JUL1240.pdf (accessed on 4 June 2022).
207. Etemadi, N.; Van Gelder, P.; Strozzi, F. An ism modeling of barriers for blockchain/distributed ledger technology adoption in supply chains towards cybersecurity. *Sustainability* **2021**, *13*, 4672. [CrossRef]
208. Wasim Ahmad, R.; Hasan, H.; Yaqoob, I.; Salah, K.; Jayaraman, R.; Omar, M. Blockchain for aerospace and defense: Opportunities and open research challenges. *Comput. Ind. Eng.* **2021**, *151*, 106982. [CrossRef]

Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges

(Tech Report$^{\mathbf{V2}}$)

Qin Wang*[2,4], Rujia Li*[1,3], Qi Wang[1], Shiping Chen[4]

[1] Southern University of Science and Technology
[2] Swinburne University of Technology
[3] University of Birmingham
[4] CSIRO Data61

Abstract. The Non-Fungible Token (NFT) market is mushrooming in recent years. The concept of NFT originally comes from a token standard of Ethereum, aiming to distinguish each token with distinguishable signs. This type of token can be bound with virtual/digital properties as their unique identifications. With NFTs, all marked properties can be freely traded with customized values according to their ages, rarity, liquidity, etc. It has greatly stimulated the prosperity of the decentralized application (DApp) market. At the time of writing (May 2021), the total money used on completed NFT sales has reached 34,530,649.86 USD. The thousandfold return on its increasing market draws huge attention worldwide. However, the development of the NFT ecosystem is still in its early stage, and the technologies of NFTs are pre-mature. Newcomers may get lost in their frenetic evolution due to the lack of systematic summaries. In this technical report, we explore the NFT ecosystems in several aspects. We start with an overview of state-of-the-art NFT solutions, then provide their technical components, protocols, standards, and desired proprieties. Afterwards, we give a security evolution, with discussions on the perspectives of their design models, opportunities and challenges. To the best of our knowledge, this is the first systematic study on the current NFT ecosystems.

Keywords: Blockchain · NFT · DApp · Smart contract

1 Introduction

Non-Fungible Token (NFT) is a type of cryptocurrency [82] that is derived by the smart contracts of Ethereum [119]. NFT was firstly proposed in Ethereum Improvement Proposals (EIP)-721 [116] and further developed in EIP-1155 [117]. NFT differs from classical cryptocurrencies [105] such as Bitcoin [98] in their intrinsic features. Bitcoin [98] is a standard coin in which all the coins are

* These authors contributed equally to the work.
Email: qinwang@swin.edu.au and rxl635@bham.ac.uk.

equivalent and indistinguishable. In contrast, NFT is unique which cannot be exchanged like-for-like (equivalently, non-fungible), making it suitable for identifying something or someone in a unique way. To be specific, by using NFTs on smart contracts (in Ethereum [119]), a creator can easily prove the existence and ownership of digital assets in the form of videos, images, arts [83], event tickets [103], etc. Furthermore, the creator can also earn royalties each time of a successful trade on any NFT market or by peer-to-peer exchanging. Full-history tradability, deep liquidity, and convenient interoperability enable NFT to become a promising intellectual property (IP)-protection solution. Although, in essence, NFTs represent little more than code, but the codes to a buyer have ascribed *value* when considering its comparative scarcity as a digital object. It well secures selling prices of these IP-related products that may have seemed unthinkable for non-fungible virtual assets.

In recent years, NFTs have garnered remarkable attention from both the industrial and scientific communities. It was reported that the 24-hour trading volume on average of the NFT market is $4,592,146,914$ USD[1], while the 24-hour trading volume of the entire cryptocurrency market is $341,017,001,809$ USD. The liquidity of NFT-related solutions has accounted for 1.3% of the entire cryptocurrency market in such a short period (5 months). Early investors obtain thousandfold returns by selling unique digital collectibles. At the time of writing (May 2021), the NFTs-related market has significantly increased compared to one year ago (January 2020). Specifically, the total number of sales is $25,729$ and their total amounts spent on completed sales reach $34,530,649.86$ USD[2]. In particular, the total number of primary-market sales occupies $17,140$, while the number of secondary sales (user-to-user) is $8,589$. Correspondingly, the total USD used on primary market sales is $8,816,531.10$. Besides, the active market wallets achieve $12,836$, which is still increasing at a high speed as time goes. Surprisingly, the sale of NFTs was estimated at 12 million (December 2020) but exploded to 340 million within just two months (February 2021). Such skyrocketing development makes NFT become a craze, or even be described by some as the future of digital assets.

Besides the above data, people have expressed interest in various types of NFTs. They participate in NFT-related games or trades with enthusiasm. CryptoPunks [20], one of the first NFT on Ethereum, has created more than $10,000$ collectible punks (6039 males and 3840 females) and further promoted the ERC-721 standard to become popular. CryptoKitties [19] officially put NFTs on notice, and hit the market in 2017 with the gamification of the breeding mechanics. Participants fiercely competed at high prices to auction the rare cats, and the highest price reaches more than 999 ETH[3] (equally 3M USD). Another outstanding instance is NBA Top Shot [44], which is an NFT trading platform used to buy/sell digital short videos of NBA moments. Thousands of NBA fans from around worldwide

[1] Data source from Coingecko (May, 2021) `https://www.coingecko.com/en/nft`
[2] Data captured from `https://nonfungible.com/market/history`
[3] Price available: `https://www.cryptokitties.co/kitty/1866420`

have collected over 7.6 million top shot moments, building the roster of rookies, vets, and rising star players. Following projects also enjoy great success including Picasso Punks [51], Hashmasks [31], 3DPunks [4], unofficial punks [63], Polkamon [52], Chubbies [14], Bullrun Babes [12], Aavegotchi [5], CryptoCats [18], Moon Cats Rescue [42], NFT box [46], etc. There is no doubt that there is a hype cycle surrounding NFTs where most of products can be sold with high prices, some even hundreds or thousands of ETHs. Besides games and collectibles, NFTs also promote the development of art [108], ticketing event [103], value [77],[78], IoT [100], and finance [80][97]. Other types of surrounding markets play important roles as well to provide instant information and secure environments like statistic websites (e.g. NonFungible [49], DappRadar [24], NFT bank[45], DefiPulse [26], Coingecko [15]), trading marketplace (cryptoslam [21], Opensea [50], SuperRare [59], Nifty Gateway [48], Rarible [55], Zora [69]) and so-called NFT ecosystem (such as Dego [25]).

Despite NFTs have a tremendous potential impact on the current decentralized markets and future business opportunities, the NFT technologies are still in the very early stage. Some potential challenges are required to be carefully tackled, while some promising opportunities should be highlighted. Further, even though much literature on NFTs, from blogs, wikis, forum posts, codes and other sources, are available to the public, a systematic study is absent. This paper aims to draw attention to these questions insofar as observed and focus on summarising current NFT solutions. We provide a detailed analysis of its core components, technology roadmap status, opportunities and challenges. The contributions are provided as follows.

- Firstly, *we abstract the design models of current NFT solutions.* Specifically, we identify the core technical components that are used to construct NFTs. Then, we present their protocols, standards and targeted properties.

- Secondly, *we give a security evaluation of current NFT systems.* We adopt the STRIDE threat and risk evaluation [106] to investigate potential security issues. Based on that, we also discuss the corresponding defense measures for the issues.

- Thirdly, *we explore some future opportunities of NFTs in many fields.* Applying NFTs to real scenarios will boost a wide range of new applications. We give a set of practical instances (projects) leveraging NFTs with great success or prosperous markets.

- Finally, *we highlight a series of open challenges in NFT ecosystems.* Blockchain-based NFT systems still confront unavoidable problems like privacy issues, data inaccessibility, etc. We outline open challenges existed in state-of-the-art NFT solutions.

The rest part of this work is organized as follows. Section 2 provides the technical components used to build NFTs. Section 3 presents the protocols and standards. Based on that, Section 4 gives our security evaluation. Section 5 discusses the

future opportunities, while Section 6 outlines the open challenges. Finally, Section 7 concludes this work. Appendix A gives our version updates. Appendix B and Appendix C provide NFT collectible ranking and an overview of existing NFT projects, respectively. Appendix D presents a detailed instance analysis.

2 Technical Components

In this part, we show technical components related to the NFT's activities. These components lay the building foundations of a fully functional NFT scheme.

Blockchain. Blockchain was originally proposed by Nakamoto [98], where Bitcoin uses the proof of work (PoW) [87] algorithm to reach an agreement on transaction data in a decentralized network. Blockchain is defined as a distributed and attached-only database that maintains a list of data records linked and protected using cryptographic protocols [86]. Blockchain provides a solution to the long-standing Byzantine problem [92], which has been agreed upon with a large network of untrusted participants. Once the shared data on the blockchain is confirmed in most distributed nodes, it becomes immutable because any changes in the stored data will invalidate all subsequent data. The most prevailing blockchain platform used in NFT schemes is Ethereum [119], providing a secure environment for executing the smart contracts. In addition, several solutions drop their customized chain-engines or blockchain platforms to support their specialized applications, and some of them are Flow [2], EOS [67], Hyperledger [89][70], and Fast Box [28][111].

Smart Contract. Smart contracts were originally introduced by Szabo [107], aiming to accelerate, verify or execute digital negotiation. Ethereum [119] further developed smart contracts in the blockchain system [84][71]. Blockchain-based smart contracts adopt Turing-complete scripting languages to achieve complicated functionalities and execute thorough state transition replication over consensus algorithms to realize final consistency. Smart contracts enable unfamiliar parties and decentralized participants to conduct fair exchanges without a trusted third party and further propose a unified method to build applications across a wide range of industries. The applications operating on top of smart contracts are based on state-transition mechanisms. The states that contain the instructions and parameters are shared by all the participants, thus guaranteeing transparency of the execution of these instructions. Also, the positions between states have to stay the same across distributed nodes, which is important to its consistency. Most NFT solutions [20][19][44][51][31][4] rely on smart contract-based blockchain platforms to ensure their order-sensitive executions.

Address and Transaction. Blockchain address and transaction are the essential concepts in cryptocurrencies. A blockchain address is a unique identifier for a user to send and receive the assets, which is similar to a bank account when spending the assets in the bank. It consists of a fixed number of alphanumeric characters generated from a pair of public key and private key. To transfer NFTs,

the owner must prove in possession of the corresponding private key and send the assets to another address(es) with a correct digital signature. This simple operation is usually performed using a cryptocurrency wallet and is represented as sending a transaction to involve smart contracts in the ERC-777 [90] standard.

Data Encoding. Encoding is the process of converting data from one form to another. Normally, many files are often encoded into either efficient, compressed formats for saving disk space or into an uncompressed format for high quality/resolution. In the mainstream blockchain systems such as Bitcoin [98] and Ethereum [119], they employ `hex` values to encode transaction elements such as the function names, parameters and return values. This implies that the raw NFT data must follow these rules. If one claims s/he owns the NFT-based intellectual property, s/he essentially owns the original piece of `hex` values signed by the creator. Others can freely copy the raw data, but they cannot claim ownership of the property. Based on that, we can observe that the NFT-related activities (e.g. buy/sell/trade/auction) have to be processed under these four phases, similar to the basic processing procedure of smart contracts.

3 Protocols, Standards and Properties

This section presents two basic models of NFT schemes, with emphasis on their protocols, token standards and key properties.

3.1 Protocols

The establishment of NFT requires an underlying distributed ledger for records, together with exchangeable transactions for trading in the peer-to-peer network. This report primarily treats the distributed ledger as a special type of database to store NFT data. In particular, we assume that the ledger has basic security consistency, completeness, and availability characteristics. Based on that, we identify two design patterns for the NFT paradigm. The former protocol is established from top to bottom with a very simple yet classical path: *building NFTs from the initiator, and then sell them to the buyer.* In contrast, the later route (e.g. Loot [34], detailed analysis in Appendix D) reverses this path: *setting a NFT template, and every user can create their unique ontop NFTs.* We separately provide detailed protocols of these two design patterns as below. To be noted, for both of them, they still follow a very similar workflow when executed on blockchain systems (cf. Fig.1), meaning that different designs will not change the underlying operating mechanism.

Top to Bottom. For the first design (e.g., CryptoPunks [20]), an NFT protocol consists of another two roles: NFT owner and NFT buyer.

- **NFT Digitize.** An NFT owner checks that the file, title, description are completely accurate. Then, s/he digitizes the raw data into a proper format.

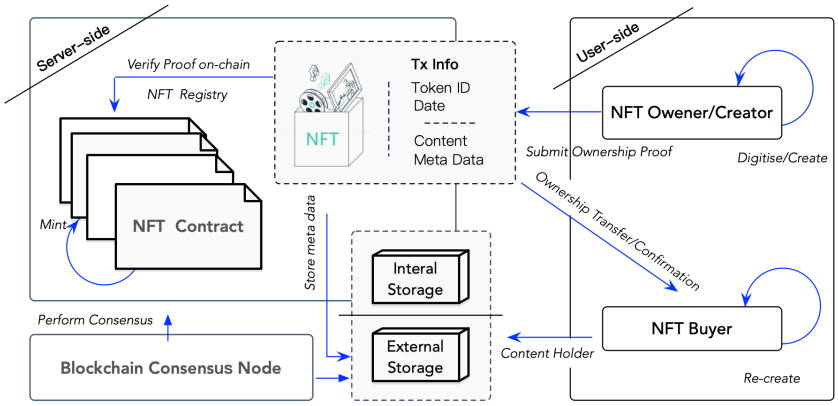


Fig. 1: Workflow of NFT Systems

- **NFT Store.** An NFT owner stores the raw data into an external database outside the blockchain. Note that, s/he is also allowed to store the raw data inside a blockchain, despite this operation is gas-consuming.
- **NFT Sign.** The NFT owner signs a transaction, including the hash of NFT data, and then sends the transaction to a smart contract.
- **NFT Mint&Trade.** After the smart contract receives the transaction with the NFT data, the minting and trading process begins. The main mechanism behind NFTs is the logic of the Token Standards that can be found in Section 3.2.
- **NFT Confirm.** Once the transaction is confirmed, the minting process completes. By this approach, NFTs will forever link to a unique blockchain address as their persistence evidence.

Bottom to Top. For this design (e.g., Loot [34]), the protocol consists of two roles: NFT creator and NFT buyer. In most cases, a buyer can also act as a creator because an NFT product is created based on random seeds when a buyer bids for it. This extends the functions in terms of user customization. Here, we use the superscript $*$ to highlight differences compared with the previous one.

- **Template Create*.** The project founder initiates a template via the smart contract to set up several basic rules, such as different features (character style, weapons, or accessories) in the game.
- **NFT Randomize*.** Once a buyer bids for an NFT, s/he can customize the NFT product with a set of additional features on top of basic lines. These additional features are randomly selected from a database that was predefined at the initial state.
- **NFT Mint&Trade.** The minting and trading process starts once the corresponding smart contract is triggered.

- **NFT Confirm.** All the procedures are conducted through smart contracts. The generated NFT will be persistently stored on-chain when the consensus procedure has been completed.

In a blockchain system, each block has a limited capacity. When the capacity in one block becomes full, other transactions will enter a future block linked to the original data block. In the end, all linked blocks have created a long-term history that remains permanent. The NFT system, in essence, is a blockchain-based application. Whenever an NFT is minted or sold, a new transaction is required to send to invoke the smart contract. After the transaction is confirmed, the NFT metadata and ownership details are added to a new block, thereby ensuring that the history of the NFT remains unchanged and the ownership is preserved.

3.2 Token Standards

In this part, we clarify token standards related to NFTs, including ERC-20 [81], ERC-721 [115], and ERC-1155 [118] (see Algorithm 1). These standards have a great impact on the ongoing NFT schemes. We discuss them as follows.

Algorithm 1: NFT Standard Interfaces (with selected functions)

```
interface ERC721 {
  function ownerOf(uint256_tokenId) external view returns (address);
  function transferFrom(address_from, address_to, uint256_tokenId)
 external payable; ...
}
interface ERC1155 {
  function balanceOf(address_owner, uint256_id) external view returns
 (address);
  function balanceOfBatch(address calldata _owners, uint256 calldata
 _ids) external view returns (uint256 memory);
  function transferFrom(address_from, address_to, uint256_id, uint256
 quantity) external payable; ...
}
```

The most prevailing token standard comes from ERC-20 [81]. It introduces the concept of fungible tokens that can be issued on top of Ethereum once satisfying the requirements. The standard makes tokens the same as another one (in terms of both type and value). An arbitrary token is always equal to all the other tokens. This stimulates the hype of Initial Coin Offering (ICO) from 2015 to present. A lot of public chains and various blockchain-based DApps [75][102] gain sufficient initial fundings in this way. In contrast, ERC-721 [115] introduces a non-fungible token standard that differs from the fungible token. This type of token is unique that can be distinguished from another token. Specifically, every

NFT has a uint256 variable called `tokenId`, and the pair of contract address and uint256 `tokenId` is globally unique. Further, the `tokenId` can be used as an input to generate special identifications such as images in the form of zombies or cartoon characters.

Another standard ERC-1155 (Multi Token Standard) [118] extends the representation of both fungible and non-fungible tokens. It provides an interface that can represent any number of tokens. In previous standards, every `tokenId` in contact only contains a single type of tokens. For instance, ERC-20 makes each token type deployed in separate contracts. As well, ERC-721 deploys the group of non-fungible tokens in a single contract with the same configurations. In contrast, ERC-1155 extends the functionality of `tokenId`, where each of them can independently represent different configurable token types. The field may contain its customized information such as the metadata, lock-time, date, supply, or any other attributes. Here, we provide an illustration (see Fig.2) to show their structures and aforementioned differences.

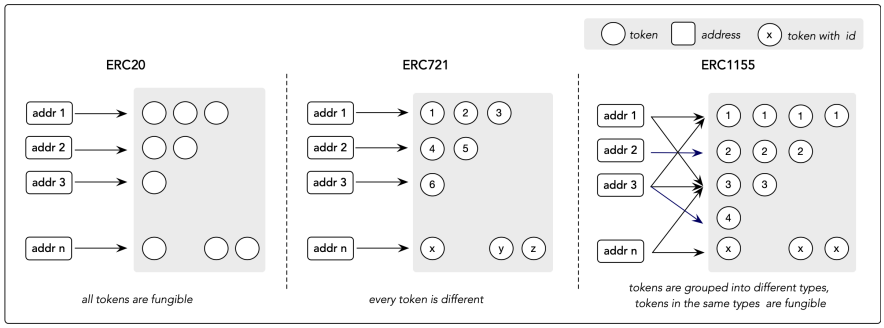


Fig. 2: NFT-related Token Standards

3.3 NFTs Desired Proprieties

NFT schemes are essentially decentralized applications [74], and thus enjoy the benefits/properties from their underlying public ledgers. We summarise the key properties as follows.

- **Verifiability.** The NFT with its token metadata and its ownership can be publicly verified.
- **Transparent Execution.** The activities of NFTs include minting, selling and purchasing are publicly accessible.
- **Availability.** The NFT system never goes down. Alternatively, all the tokens and issued NFTs are always available to sell and buy.

- **Tamper-resistance.** The NFT metadata and its trading records are persistently stored and cannot be manipulated once the transactions are deemed as confirmed.
- **Usability.** Every NFT has the most up-to-date ownership information, which is user-friendly and information-clearly.
- **Atomicity.** Trading NFTs can be completed in one atomic, consistent, isolated, and durable (ACID) transaction. The NFTs can run in the same shared execution state.
- **Tradability.** Every NFTs and its corresponding products can be arbitrarily traded and exchanged.

4 Security Evaluation

An NFT system is a combination technology that consists of blockchain, storage and web application. Security evaluation on the NFT system is challenging since each component may become an attacking interface that makes the whole system really vulnerable against the attacker. Thus, we adopt the STRIDE threat and risk evaluation [106], which covers all security aspects of a system: authenticity, integrity, non-repudiability, availability and access control. We investigate the potential security issues and propose some of the corresponding defense measures to address these issues (see Table 1).

Spoofing. Spoofing is the ability to impersonate another entity (for example, another person or computer) on the system, which corresponds to authenticity. When a user interacts to mint or sells NFTs, a malicious attacker may exploit authentication vulnerabilities or steal the user's private key to transfer the ownership of NFTs illegally. Thus, we recommend having a formal verification for the NFT smart contract and to use the cold wallet to prevent private key leakage.

Tampering. Tampering refers to the malicious modification of NFT data, which violates integrity. Assume that the blockchain is a *robust* public transaction ledger [85,86] and a hash algorithm is preimage resistance and second preimage resistance [104]. The metadata and ownership of NFTs cannot be maliciously modified after the transaction is confirmed. However, the data stored outside blockchain may be manipulated. Therefore, we recommend users to send both the hash data as well as the original data to the NFT buyer when trading/exchanging NFT-related properties.

Repudiation. Repudiation refers to the situation where the author of a statement cannot dispute [121], which is related to the security property of non-repudiability [95]. In particular, the fact that a user sends NFT to another user cannot deny. This is guaranteed by the security of the blockchain and the unforgeability property of a signature scheme. However, the hash data may be tampered by a malicious attacker, or the hash data may bind with an attacker's

address. Thus, we believe that using a multi-signature contract can partly solve this issue since each binding must be confirmed by more than one participant.

Information Disclosure. Information leakage occurs when information is exposed to unauthorized users, which violates confidentiality [95]. In the NFT system, the state information and the instruction code in the smart contracts are entirely transparent, and any state and its changes are publicly accessible by any observer. Even if the user only puts the NFT hash into the blockchain, the malicious attackers can easily exploit the linkability of the hash and transaction. Thus, we recommend the NFT developer to use privacy-preserving smart contracts [93][94] instead of plain smart contracts to protect the user's privacy.

Table 1: Potential Security Issues and Corresponding Solutions of NFTs.

STRIDE	Security Issues	Solutions
Spoofing *(Authenticity)*	• An attacker may exploit authentication vulnerabilities • An attacker may steal a user's private key.	• A formal verification on the smart contract. • Using the cold wallet to prevent the private key leakage.
Tampering *(Integrity)*	• The data stored outside the blockchain may be manipulated.	• Sending both the original data and hash data to the NFT buyer when trading NFTs.
Repudiation *(Non-repudiability)*	• The hash data may bind with an attacker's address.	• Using a multi-signature contract partly.
Information disclosure *(Confidentiality)*	• An attacker can easily exploit the hash and transaction to link a particular NFT buyer or seller.	• Using privacy-preserving smart contracts instead of smart contracts to protect the user's privacy.
Denial of service *(Availability)*	• The NFT data may become unavailable if the asset is stored outside the blockchain.	• Using the hybrid blockchain architecture with weak consensus algorithm.
Elevation of privilege *(Authorization)*	• A poorly designed smart contract may make NFTs lose such properties.	• A formal verification on the smart contracts.

Denial of Service (DoS). DoS attack [96] is a type of network attack in which a malicious attacker aims to render a server unavailable to its intended users by interrupting the normal functions. DoS violates the availability and breaks down the NFT service, which can indeed be used by unauthorized users. Fortunately, the blockchain guarantees the high availability of user's operations.

Legitimate users can use the required information when needed and will not lose data resources due to accidental errors. However, DoS can also be used to attack the centralized web applications or the raw data outside the blockchain, resulting in denial-of-service to NFT service. Recently, a new hybrid blockchain architecture with weak consensus algorithm was proposed [111], by which this architecture solves the availability issues using two algorithms.

Elevation of Privilege. Elevation of Privilege [106] is a property that is related to the authorization. In this type of threat, an attacker may gain permissions beyond those initially granted. In the NFT system, the selling permissions are managed by a smart contract. Again, a poorly designed smart contract may make NFTs lose such properties.

5 Opportunities

This section explores the opportunities of NFTs. We discuss several typical fields which may get benefits from NFTs.

Boosting Gaming Industry. NFT has great potential in the gaming industry. There already exist some crypto games are CrytpoKitties [19], Cryptocats [18], CryptoPunks [20], Meebits [38], Axie Infinity [9], Gods Unchanged [29], and TradeStars [61]. A fascinating feature of such games is the "breeding" mechanism. Users can personally raise pets and spend much time breeding new offspring. They can also purchase the limited/rare edition virtual pets, and then sell them at a high price. The extra reward attracts lots of investors to join the games, making NFTs come to prominence. Another exciting functions of the NFT are that it provides ownership records of items in the games and promotes economic marking place in the ecosystem, benefiting both developers and players. In particular, game developers who are NFT publishers of the features (e.g., weapons and skins) can earn royalties each time their items are (re-)sold on the open market. The players can obtain personal exclusivity game items. This will create a mutually beneficial business model in which both players and developers profit from the secondary NFT market. After that, blockchain communities extend NFTs to a large extent that covers various types of digital assets.

Flourishing Virtual Events. Traditional online events rely on centralized companies that provide trust and technology. Although blockchain takes over several types of activities like raising money (either by ICO/IFO/IEO/etc.), its applications are still constrained in a small range of events. NFTs greatly extend the scope of blockchain applications with the help of their additional properties (uniqueness, ownership, liquidity). This enables each individual to link to a specific event just like the patterns in our real life. We give the instance of the ticketing event. When buying tickets in a traditional event ticket market, consumers must trust the third party. Therefore, there is a risk of buying fraudulent or invalid tickets, which are possibly counterfeit or might be cancelled. The same ticket may be sold many times or obtained by extracting from ticket

images posted online in an extreme case. "NFT-based ticket" represents a ticket issued by the blockchain to demonstrate entitlement to access to any event such as culture or sports. An NFT-based ticket is unique and scarce, meaning that the ticket holder cannot resell the ticket after it is sold. The blockchain-based smart contract provides a transparent ticket trading platform for the stakeholders such as the event organizer and the customer. Consumers can buy and sell the crypto ticket from the smart contract rather than rely on third parties in an efficient and reliable way.

Protecting Digital Collectibles. Digital collectibles contain a variety of types, ranging from trading cards, wines [68], digital images [59], videos [44], virtual real estate [1], domain names [27][64], diamonds [32][47], crypto stamps [17] and other real/intellectual properties. We take the field of arts as an example. Firstly, artists in traditional ways have very few channels to display the works. The prices cannot reflect the true value of their works due to the absence of attention. Even worse, their published work on social networks has been charged with intermediary fees by platforms and advertisements. NFTs transform their work into digital formats with integrated identities. Artists do not have to transfer ownership and contents to agents. This provides them impetus with lots of profits. Typical instances include Mad Dog Jones's REPLICATOR (selling with 4.1 million USD [36]), Grimes's work (selling in total around 6 million USD [30]) and other works from great crypto-artists like Beeple [10] / Trevor Jones [62]. Furthermore, artists in general cases cannot receive royalties from future sales of their works. In contrast, NFTs can be programmed so that the artist receives a predetermined royalty fee each time when his digital artwork exchanges in the markets (e.g. [33], SuperRare [59], MakersPlace [37], Rare Art Lab[54], VIV3 [65]). This is an efficient way to manage and protect digital masterpieces. In addition, several platforms (e.g. Mintbase [41], Mintable [40]) have even established tools to support ordinary people to create their own NFT works easily.

Inspiring the Metaverse. Metaverse is a collective virtual shared space that allows all types of digital activities. Generally, it covers a set of techniques like augmented reality and the Internet to establish the virtual world. The concept stems from the last decades and has a great progress with the rapid development of blockchain. Blockchain provides an ideal decentralized environment for the virtual online world. Participants under this blockchain-fueled alternative realities [50] can have many types of intriguing use cases like enjoying games, displaying self-made arts, trading assets and virtual properties (arts, land parcels, names, video shots, wearables), etc. In addition, users also have opportunities to get profits from the virtual economy. They can lease the buildings (such as offices) to others to earn the bond or raise rare pets and sell them to get the rewards. Primary blockchain-empowered projects are Decentraland [1], Cryptovoxels [22], Somnium Space [57], MegaCryptoPolis [39] and Sandbox [3]. In fact, the metaverse ecosystem covers all aforementioned applications. We list it separately here simply because it is still in an early stage due to the complexity.

6 Challenges

To enable the development of the above NFT applications, a series of barriers have to be overcome as with any nascent technologies. We discuss some typical challenges from the perspectives of usability, security, governance, and extensibility, covering both the system level issues caused by blockchain-based platforms and human factors such as governs, regulation, and society.

6.1 Usability Challenges

Usability is to measure the users' effectiveness, efficiency, and satisfaction when testing a specific product/design. Most NFT schemes are built on top of Ethereum. Therefore, it is obvious that the main drawbacks of Ethereum still exist. We discuss two major challenges that have direct impacts on the user experience.

- **Slow Confirmation.** NFT-related procedures are tyically conducted by sending transactions via the smart contract for reliable and transparent management (such as mint, sell, exchange). However, current NFT systems are closely coupled with their underlying blockchain platforms, which makes them suffer from low performance (Bitcoin reaches merely 7 TPS [109] while Ethereum only 30 TPS). This results in extremely slow confirmation of NFTs. Conquering this issue requires a redesign of blockchain systems [113], optimization of its structure [110][88] or improvement on the consensus mechanisms [71]. Existing blockchain systems cannot fulfil such requirements.
- **High Gas Prices.** High gas prices have become a major problem for NFT marketplaces, especially when minting the NFTs at a large scale that requires uploading the metadata to the blockchain network. Every NFT-related transactions are more expensive than a simple transfer transaction because smart contracts involve computational resources and storage to be processed. At the time of writing, to mine an NFT token costs over USD 60 (equivalently in around $5 \times 10^2 wei$) [1]. To complete a simple NFT trade can run between USD 60 and USD 100 for each transaction. Expensive fees caused by complex operations and high congestion greatly limit its wide adoption.

6.2 Security and Privacy Issues

The security of user data pose the first priority of systems. However, the data (stored off-chain but relates to on-chain tags) confronts the risk of losing linkage or being misused by malicious parties. We give details as follows.

- **NFT Data Inaccessibility.** In the mainstream NFT projects [2,1,3], a cryptographic "hash" as the identifier, instead of a copy of the file, will be tagged with the token and then recorded on the blockchain to save the gas

[1] Source calculated from `https://coinmarketcap.com/` and `https://ethereum.org/en/developers/docs/gas/`

consumption. This makes the user lose confidence in the NFT because the original file might be lost or damaged. Several NFT projects integrate their system with a specialized file storage system such as IPFS [72] in which IPFS addresses allow users to find a piece of content so long as someone somewhere on the IPFS network is hosting it. Inevitably, such systems have flaws. When the users "upload" NFT metadata to IPFS nodes, there is no guarantee that their data will be replicated among all the nodes. The data may become unavailable if the asset is stored on IPFS and the only node storing it is disconnected from the network [73]. This issue has been reported by DECRYPT.IO [73] and CHECKMYNFT.COM [76]. Also, an NFT might point to an erroneous file address. If that is the case, a user cannot prove that s/he actually owns the NFT. In a word, relying on an external system as the core component (storage) for an NFT system is vulnerable.

- **Anonymity/Privacy.** In the current stage, the anonymity and privacy of NFTs are still understudied. Most NFT transactions rely on their underlying Ethereum platform, which only provides pseudo-anonymity rather than strict anonymity or privacy. Users can partially hide their identities if the links between their real identities and corresponding addresses are unknown by the public. Otherwise, all the activities of users under the exposed address are observable. Existing privacy-preserving solutions (e.g. homomorphic encryption [112], zero-knowledge proof [114], ring signature [99], multi-party computation [101]) have not been yet applied to the NFT-related schemes due to their complicated cryptographic primitives and security assumptions. Similar to other types of blockchain-based systems, decreasing expensive computation costs becomes the key to implement privacy-promised schemes.

6.3 Governance Consideration

Similar to the situations of most cryptocurrencies, NFTs also confront the barriers like strict management from the government. On the other side, how to properly regulate this nascent technology with the corresponding market is also a challenge. We discuss two typical issues from both sides.

- **Legal Pitfalls.** NFTs confront legal and policy issues across a wide range of areas [82][91]. Potential concerned areas cover commodities, cross-border transactions, KYC (Know Your Customer) data, etc. It is important to understand the related regulatory scrutiny and litigation before moving into the NFT tracks. In some countries, such as Indian and China, the legal situation is strict for cryptocurrencies, and also for NFT sales. Exchanging, trading, selling, or buying NFTs have to overcome the difficulties of governance. Legally, users can only trade derivates on authorized exchanges such as stocks and commodities or exchange tokens with someone person-to-person. Several countries, such as Malta and France, are trying to implement suitable laws with the aim to regulate the service of digital assets. Elsewhere, issues are resolved by using existing laws. They require buyers to follow com-

plex or even contradictory terms. Therefore, undertaking due diligence is a necessity before investing serious tokens in NFTs.

- **Taxable property Issues.** IP-related products (including arts, books, domain names, etc.) are treated as taxable property under the current legal framework. However, NFT-based sales stay out of this scope. Although few countries, such as the U.S. (internal revenue service, IRS), tax cryptocurrencies as property, most areas worldwide have not yet considered it. This may greatly increase the financial crimes under cover of NFT trading. The governments would love to make the sale of NFTs reliable with tax consequences. Specifically, the individual participants should have the tax liability on any capital gains that are related to NFT properties. Also, NFT-for-NFT, NFT-for-IP, and Eth-for-NFT (or vice versa) exchanges should be taxed. Furthermore, for high-profit properties, or collectibles, a higher tax bracket should be applied. Thus, NFT-related trades are suggested to seek more advice from professional tax departments after the profound discussions.

6.4 Extensibility Issues

The extensibility of NFT schemes is two-fold. The first is to stress whether a system can interact with other ecosystems. The second focuses on whether NFT systems can obtain updates when the current version is left behind.

- **NFT Interoperability (cross-chain).** Existing NFT ecosystems are isolated from each other. Users once have selected one type of product can only sell/buy/trade them within the same ecosystem/network. This is due to the reason of its underlying blockchain platform. Interoperability and cross-chain communication are always the handicaps for the wide adoption of DApps. Based on the observations from [120], cross-chain communications can only be implemented with the help of external trusted parties. The decentralization property, in this way, has been inevitably lost to some extent. But fortunately, most of the NFT-related projects adopt Ethereum as their underlying platform. This indicates that they share a similar data structure and can exchange under the same rules.
- **Updatable NFTs.** Transitional blockchains update their protocols through soft forks (minor modifications that are compatible forwards) and hard forks (significant modifications that may conflict with previous protocols). A formal discussion has been provided in [79] stating the difficulties and tradeoffs when applying the updates to an existing blockchain. Despite using the generic model, a new version still faces strict requirements such as tolerating specific adversarial behaviours and staying online during the update process. NFT schemes closely rely on their underlying platforms and keep consistent with them. Although the data are often stored in separate components (such as the IPFS file system), the most important logic and tokeId are still recorded on-chain. Properly updating the system with improvements will be a necessity.

7 Conclusion

Non-Fungible Token (NFT) is an emerging technology prevailing in the blockchain market. In this report, we explore the state-of-the-art NFT solutions which may re-shape the market of digital/virtual assets stepping forward. We firstly analyze the technical components and provide the design models and properties. Then, we evaluate the security of current NFTs systems and further discuss the opportunities and potential applications that adopt the NFT concept. Finally, we outline existing research challenges that require to be solved before achieving mass-market penetration. We hope this report delivers timely analysis and summary of existing proposed solutions and projects, making it easier for newcomers to keep up with the current progress.

References

1. Decentraland (mana). Project accessible: `https://decentraland.org/` (2020)
2. Flow. Project accessible: `https://www.onflow.org/` (2020)
3. Sandbox. Project accessible: `https://www.sandbox.game/en/` (2020)
4. 3d punks. Project Accessible: `http://www.3dpunks.com/` (2021)
5. Aavegotchi. Project Accessible: `https://aavegotchi.com/` (2021)
6. Alien worlds. Project Accessible: `https://alienworlds.io/` (2021)
7. Art blocks. Project Accessible: `https://artblocks.io/` (2021)
8. Async art. Project Accessible: `https://async.art/` (2021)
9. Axie infinity. Project Accessible: `https://axieinfinity.com/` (2021)
10. Beeple. Project Accessible: `https://www.beeple-crap.com/` (2021)
11. Bored ape yacht club. Project Accessible: `https://boredapeyachtclub.com/#/` (2021)
12. Bullrun babes. Project Accessible: `https://opensea.io/collection/bullrunbabestoken` (2021)
13. Cargo. Project Accessible: `https://cargo.build/` (2021)
14. Chubbies. Project Accessible: `https://chubbies.io/` (2021)
15. Coingecko website. Accessible: `https://coingecko.com/en` (2021)
16. Cometh. Project Accessible: `https://www.cometh.io/` (2021)
17. Crypto stamp. Project Accessible: `https://crypto.post.at/` (2021)
18. Cryptocats. Project Accessible: `https://cryptocats.thetwentysix.io/` (2021)
19. Cryptokitties. Project Accessible: `https://www.cryptokitties.co/` (2021)
20. Cryptopunks. Accessible: `https://www.larvalabs.com/cryptopunks` (2021)
21. Cryptoslam. Project Accessible: `https://cryptoslam.io/` (2021)
22. Cryptovoxels. Project Accessible: `https://www.cryptovoxels.com/` (2021)
23. Cryptowine. Project Accessible: `https://grap.finance/#/` (2021)
24. Dappradar website. Accessible: `https://dappradar.com/` (2021)
25. Dego. Project Accessible: `https://dego.finance/` (2021)
26. Deipulse website. Accessible: `https://defipulse.com/` (2021)
27. Ens domains. Project Accessible: `https://app.ens.domains/` (2021)
28. Fast box. Project Accessible: `https://www.fastbox.cc/` (2021)
29. Gods unchanged. Project Accessible: `https://godsunchained.com/` (2021)
30. Grimes sold $6 million worth of digital art as nfts. News source: `https://www.theverge.com/2021/3/1/22308075/grimes-nft-6-million-sales-nifty-gateway-warnymph` (2021)

31. Hashmasks. Project Accessible: https://www.thehashmasks.com/ (2021)
32. Icecap. Project Accessible: https://icecap.diamonds/ (2021)
33. Known origin. Project Accessible: https://www.knownorigin.io/ (2021)
34. Loot contract code. https://etherscan.io/address/0xff9c1b15b16263c61d017ee9f65c50e4ae0113d7#code (2021)
35. Loot talk. https://loot-talk.com/ (2021)
36. Mad dog jones. News source: https://www.phillips.com/detail/mad-dog-jones/NY090121/1 (2021)
37. Makersplace. Project Accessible: https://makersplace.com/ (2021)
38. Meebits. Project Accessible: https://meebits.larvalabs.com/ (2021)
39. Megacryptopolis. Project Accessible: https://mcp3d.com/ (2021)
40. Mintable. Project Accessible: https://mintable.app/ (2021)
41. Mintbase. Project Accessible: https://www.mintbase.io/ (2021)
42. Moon cats rescue. Project Accessible: https://mooncatrescue.com/ (2021)
43. Mycryptoheroes. Project Accessible: https://www.mycryptoheroes.net/ (2021)
44. Nba top shot. Accessible: https://nbatopshot.com/ (2021)
45. Nft bank. Project Accessible: https://nftbank.ai/ (2021)
46. Nft box. Project Accessible: https://nftboxes.io/ (2021)
47. Nft diamonds. Project Accessible: https://nftdiamonds.co/ (2021)
48. Nifty gateway. Project Accessible: https://niftygateway.com/ (2021)
49. Nonfungible website. Accessible: https://NonFungible.com (2021)
50. Opensea platform. Accessible: https://opensea.io/ (2021)
51. Picasso punks. Accessible: https://opensea.io/collection/picassopunks (2021)
52. Polkamon. Project Accessible: https://polkamon.com/ (2021)
53. R planet. Project Accessible: https://rplanet.io/ (2021)
54. R.a.r.e art lab. Project Accessible: https://www.lagelnd.com/rare (2021)
55. Rarible. Project Accessible: https://rarible.com/ (2021)
56. Skyweaver. Project Accessible: https://www.skyweaver.net/ (2021)
57. Somnium space. Project Accessible: https://somniumspace.com/ (2021)
58. Sorare. Project Accessible: https://sorare.com/ (2021)
59. Superrare. Project Accessible: https://superrare.co/ (2021)
60. Topps mlb. Project Accessible: https://toppsmlb.com/ (2021)
61. Tradestars. Project Accessible: https://tradestars.app/ (2021)
62. Trevorjonesart. Project Accessible: https://www.trevorjonesart.com/ (2021)
63. Unofficial punks. Project Accessible: https://opensea.io/collection/unofficialpunks (2021)
64. Unstoppable domains. Accessible: https://unstoppabledomains.com/ (2021)
65. Viv3. Project Accessible: https://VIV3.com (2021)
66. Wax: Worldwide asset exchange. Accessible: https://on.wax.io/wax-io/ (2021)
67. Wax:worldwide asset exchange. Project Accessible: https://github.com/worldwide-asset-exchange/whitepaper (2021)
68. Wiv. Project Accessible: https://www.wiv.io/ (2021)
69. Zora. Project Accessible: https://zora.co/ (2021)
70. Bal, M., Ner, C.: Nftracer: a non-fungible token tracking proof-of-concept using hyperledger fabric. arXiv preprint arXiv:1905.04795 (2019)
71. Bano, S., Sonnino, A., Al-Bassam, M., Azouvi, S., McCorry, P., Meiklejohn, S., Danezis, G.: Sok: Consensus in the age of blockchains. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 183–198 (2019)
72. Benet, J.: Ipfs-content addressed, versioned, p2p file system. arXiv preprint arXiv:1407.3561 (2014)

73. Benson, J.: Your nfts can go missing—here's what you can do about it. https://decrypt.co/62037/missing-or-stolen-nfts-how-to-protect (2021)
74. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. white paper **3**(37) (2014)
75. Cai, W., Wang, Z., et al.: Decentralized applications: The blockchain-empowered software system. IEEE Access **6**, 53019–53033 (2018)
76. CH21: Check my nft. https://checkmynft.com/ (2021)
77. Chevet, S.: Blockchain technology and non-fungible tokens: Reshaping value chains in creative industries. Available at SSRN 3212662 (2018)
78. Chohan, U.W.: Non-fungible tokens: Blockchains, scarcity, and value. Critical Blockchain Research Initiative (CBRI) Working Papers (2021)
79. Ciampi, M., et al.: Updatable blockchains. In: European Symposium on Research in Computer Security. pp. 590–609. Springer (2020)
80. Dowling, M.M.: Fertile land: Pricing non-fungible tokens. Available at SSRN 3813522 (2021)
81. Fabian, V., Vitalik, B.: Eip-20: Erc-20 token standard. Accessible: https://eips.ethereum.org/EIPS/eip-20 (2015)
82. Fairfield, J.: Tokenized: The law of non-fungible tokens and unique digital property. Indiana Law Journal, Forthcoming (2021)
83. Franceschet, M., Colavizza, G., Smith, T., et al.: Crypto art: A decentralized view. Leonardo pp. 1–8 (2020)
84. Garay, J., Kiayias, A.: Sok: A consensus taxonomy in the blockchain era. In: Cryptographers' Track at the RSA Conference. pp. 284–318. Springer (2020)
85. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT). pp. 281–310. Springer (2015)
86. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty. In: CRYPTO. pp. 291–323. Springer (2017)
87. Gervais, A., et al.: On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 3–16. ACM (2016)
88. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: Sok: Layer-two blockchain protocols. In: International Conference on Financial Cryptography and Data Security. pp. 201–226. Springer (2020)
89. Hong, S., Noh, Y., Park, C.: Design of extensible non-fungible token model in hyperledger fabric. In: Proceedings of the 3rd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers. pp. 1–2 (2019)
90. Jacques, D., Jordi, B., Thomas, S.: Eip-777: Erc-777 token standard. Accessible: https://eips.ethereum.org/EIPS/eip-777 (2017)
91. Johnson, K.N.: Decentralized finance: Regulating cryptocurrency exchanges. William & Mary Law Review **62** (2021)
92. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. In: Concurrency: the Works of Leslie Lamport, pp. 203–226 (2019)
93. Li, R., Galindo, D., Wang, Q.: Auditable credential anonymity revocation based on privacy-preserving smart contracts. In: Data Privacy Management, Cryptocurrencies and Blockchain Technology, pp. 355–371. Springer (2019)
94. Li, R., et al.: An accountable decryption system based on privacy-preserving smart contracts. In: International Conference on Information Security. pp. 372–390. Springer (2020)
95. Menezes, A.J., Van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC press (2018)

96. Moore, D., Voelker, G., Savage, S.: Inferring internet denial-of-service activity. ACM Trans. Comput. Syst. **24**, 115–139 (2006)
97. Musan, D.I., William, J., Gervais, A.: Nft. finance leveraging non-fungible tokens (2020)
98. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep., Manubot (2019)
99. Noether, S.: Ring signature confidential transactions for monero. IACR Cryptol. ePrint Arch. **2015**, 1098 (2015)
100. Omar, A.S., Basir, O.: Capability-based non-fungible tokens approach for a decentralized aaa framework in iot. In: Blockchain Cybersecurity, Trust and Privacy, pp. 7–31. Springer (2020)
101. Raman, R.K., et al.: Trusted multi-party computation and verifiable simulations: A scalable blockchain approach. arXiv preprint arXiv:1809.08438 (2018)
102. Raval, S.: Decentralized applications: harnessing Bitcoin's blockchain technology. " O'Reilly Media, Inc." (2016)
103. Regner, F., Urbach, N., Schweizer, A.: Nfts in practice–non-fungible tokens as core component of a blockchain-based event ticketing application (2019)
104. Rogaway, P., Shrimpton, T.: Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: FSE. pp. 371–388. Springer (2004)
105. Shirole, M., Darisi, M., Bhirud, S.: Cryptocurrency token: An overview. IC-BCT 2019 pp. 133–140 (2020)
106. Shostack, A.: Experiences threat modeling at microsoft. MODSEC@ MoDELS **2008** (2008)
107. Szabo, N.: Smart contracts: building blocks for digital markets. EXTROPY: The Journal of Transhumanist Thought,(16) **18**(2) (1996)
108. Trautman, L.J.: Virtual art and non-fungible tokens. Available at SSRN 3814087 (2021)
109. Valdeolmillos, D., et al.: Blockchain technology: a review of the current challenges of cryptocurrency. In: International Congress on Blockchain and Applications. pp. 153–160. Springer (2019)
110. Wang, G., et al.: Sok: Sharding on blockchain. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 41–61 (2019)
111. Wang, Q., Li, R.: A weak consensus algorithm and its application to high-performance blockchain. In: IEEE INFOCOM 2021-IEEE Conference on Computer Communications (INFOCOM). IEEE (2021)
112. Wang, Q., Qin, B., Hu, J., Xiao, F.: Preserving transaction privacy in bitcoin. Future Generation Computer Systems **107**, 793–804 (2020)
113. Wang, Q., Yu, J., Chen, S., Xiang, Y.: Sok: Diving into dag-based blockchain systems. arXiv preprint arXiv:2012.06128 (2020)
114. Wang, Y., Kogan, A.: Designing confidentiality-preserving blockchain-based transaction processing systems. International Journal of Accounting Information Systems **30**, 1–18 (2018)
115. William, E., Dieter, S., Jacob, E., Nastassia, S.: Eip-721: Erc-721 non-fungible token standard. Accessible: `https://eips.ethereum.org/EIPS/eip-721` (2018)
116. William, E., Dieter, S., Jacob, E., Nastassia, S.: Erc-721 non-fungible token standard. Ethereum Improvement Protocol, EIP-721, Accessible: `https://eips.ethereum.org/EIPS/eip-721`. (2018)
117. Witek, R., Andrew, C., Philippe, C., James, T., Eric, B., Ronan, S.: Eip-1155: Erc-1155 multi token standard. Ethereum Improvement Protocol, EIP-1155, Accessible: `https://eips.ethereum.org/EIPS/eip-1155`. (2018)

118. Witek, R., et al.: Eip-1155: Erc-1155 multi token standard. Accessible: https://eips.ethereum.org/EIPS/eip-1155 (2018)
119. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)
120. Zamyatin, A., et al.: Sok: communication across distributed ledgers. (2019)
121. Zhou, J., Gollman, D.: A fair non-repudiation protocol. In: Proceedings 1996 IEEE Symposium on Security and Privacy. pp. 55–61. IEEE (1996)

Appendix A. Version Control

This work is an ongoing updated technique report. In the future, we will add the landmark events from *in the wild* NFT projects to inspire technique movements and developments. Dynamic data (like prices, sales, or market cap) will remain as it is in the current snapshot [as of May 2021].

Table 2: Version Updates

Version	Date	Updates
V1	May 2021	Main body construction
V2	October 2021	Adding the *bottom to top* model in Section 3
		Adding the instance analysis of the Loot project in Appendix D
		Revisions of unclear sentences and paragraphs

Appendix B. Top Sales of NFT Properties

Table 3: NFT Collectible Ranking by Sales Volume (All-time)

Rank	Product	Sales	Buyers	Txns	Owners
1	NBA Top Shot [44]	$573,815,060.38	285,306	4,877,975	498,659
2	CryptoPunks [20]	$316,956,115.15	2,716	13,231	2,282
3	Meebits [38]	$54,750,807.84	1,336	2,995	4,436
4	Hashmasks [31]	$49,713,202.76	3,165	11,322	4,254
5	Sorare [58]	$39,807,257.19	16,435	225,909	18,050
6	CryptoKitties [19]	$33,230,422.66	100,624	762,562	
7	Art Blocks [7]	$18,824,119.18	2,052	10,907	4,521
8	Alien Worlds [6]	$17,239,575.21	165,818	2,280,968	1,124,901
9	Bored Ape Yacht Club [11]	$11,844,235.71	2,281	5,786	2,697
10	Topps MLB [60]	$10,205,104.07	12,706	365,963	36,283

Data captured on May 10, 20201, source from: https://cryptoslam.io/

Appendix C. Overview of Existing NFT solutions

Table 4: Existing NFTs Projects (May, 2021)

Types	Projects	Platform	Key Words
Collectible	NBA Top Shot [44]	Flow [2]	Basketball moments
	CryptoPunks [20]	Ethereum	The first NFT products
	CryptoWine [23]	Ethereum	Mining by graps / Trading wines
	Hashmask [31]	Ethereum	Digital arts / Blind boxes
Cards	Sorare [58]	Ethereum	Football star cards
	Skyweaver [56]	Ethereum	Card game
	Cometh [16]	Ethereum	Galaxy background
	Gods Unchained [29]	Ethereum	Trading card game / Hearthstone
	Topps MLB [60]	WAX [66]	Baseball cards
	R Planet [53]	WAX	Red planet background
Raising Pets	CryptoKitties [19]	Ethereum	The first game with hype in Ethereum
	CryptoCats [18]	Ethereum	Raising pets and sell with high prices
	Axie Infinity [9]	Ethereum	
Virtual World	Sandbox [3]	Ethereum	Create, explore and trade in the virtual world owned by users, also make money by selling the properties.
	Decentraland [1]	Ethereum	
	Sommnium Space [57]	Ethereum	
	Cryptovoxels [22]	Ethereum	
	Alien Worlds [6]	WAX	
	MyCryptoHeros [43]	Ethereum	Historical story / Japan made
Real Properties	Crypto stamp [17]	Ethereum	NFTs on stamps
	Diamonds [47]	Ethereum	NFTs on diamonds
	Icecap [32]	Ethereum	
	Wiv [68]	Ethereum	NFTs on wine
Art Market	SuperRare [59]	-	A marketplace to create, collect and trade unique, single-edition artworks. Every artwork is authentically created by artists, and tokenized as a crypto-collectible digital item.
	Rarible [55]	-	
	Cargo [13]	-	
	Async Art [8]	-	
	Nifty Gateway [48]	-	
	KnownOrigin [33]	-	
Statistic Web	NonFungible [49]	-	Trace and monitor NFT-related projects or generic types of cryptocurrencies by providing timely statistical data to show trends.
	DappRadar [24]	-	
	NFT bank[45]	-	
	DefiPulse [26]	-	
	Coingecko [15]	-	
Exchanges	Cryptoslam [21]		A marketplace to trade unique, single-edition NFT-based properties.
	Opensea [50]	-	
	Zora [69])	-	

Sources from NonFungible, Cryptoslam, DefiPulse, Coingecko, and DappRadar.

Appendix D. Analysis of Loot

Loot [34][35] is a type of NFT product that uses ERC protocol on Ethererum, with totally 8,000 entries (7778 for trading, 222 for team incentives). The project presents all their products in the form of TEXT, removing functionalities of images, stats, or any representative formats. Users can only see eight lines of sentences, describing the attributes of game gears (e.g., weapons, armor, necklaces, Rings). In order to create scarcity, Loot adds randomized prefixes (with 42% probability) or suffixes (8.7%) for each line of attributes (see the following code, line 2), according to the assigned tokenID from users. Users can only use Etherscan's smart contract to cast loot, rather than its official website.

Selected code in Loot project

```
1  function pluck(tokenId, keyPrefix, sourceArray) internal view returns (string
       memory) {
2      uint256 rand = random(string(abi.encodePacked(keyPrefix, toString(tokenId
           ))));
3      string memory output = sourceArray[rand % sourceArray.length];
4      uint256 greatness = rand % 21;
5      if (greatness > 14) {
6          output = string(abi.encodePacked(output, " ", suffixes[rand %
               suffixes.length]));
7      }
8      if (greatness >= 19) {
9          string[2] memory name;
10         name[0] = namePrefixes[rand % namePrefixes.length];
11         name[1] = nameSuffixes[rand % nameSuffixes.length];
12         if (greatness == 19) {
13             output = string(abi.encodePacked('"', name[0], ' ', name[1], '" '
                   , output));
14         } else {
15             output = string(abi.encodePacked('"', name[0], ' ', name[1], '" '
                   , output, " +1"));
16         }
17     }
18     return output;
19 }
```

As faced by other NFTs, Loot confronts many drawbacks either. Firstly, the value of NFT is uncertain. Loot is still in the early stage of development in the current stage. Despite the price continuing to rise, there is no lack of hype. The real consensus from communities should be maintained for a long period. Secondly, difficulty in minting and high gas cost retard its wide participant. Loot does not have a front-end Dapps, which can only be minted in the Web3 browser. Complex steps become a big challenge for newcomers. Meanwhile, the high gas fee of operations in Ethereum keeps many people away from this game. The trading or minting fees can reach up to tens of hundreds of dollars for each operation. Thirdly, the extensibility needs further improvements. The current version (experimental stage) merely covers eight explicit attributes in the RPG game. Also, there is no space for on-top applications. The only future might be to develop more attributes and slots for sale.

An Overview of Cryptography

Manuel Sancha

An Overview of Cryptography

Gary C. Kessler
16 April 2015

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

Does increased security provide comfort to paranoid people? Or does security provide some very basic protections that we are naive to believe that we don't need? During this time when the Internet provides essential communication between tens of millions of people and is being increasingly used as a tool for commerce, security becomes a tremendously important issue to deal with.

There are many aspects to security and many applications, ranging from secure commerce and payments to private communications and protecting passwords. One essential aspect for secure communications is that of cryptography, which is the focus of this chapter. But it is important to note that while cryptography is *necessary* for secure communications, it is not by itself *sufficient*. The reader is advised, then, that the topics covered in this chapter only describe the first of many steps necessary for better security in any number of situations.

This paper has two major purposes. The first is to define some of the terms and concepts behind basic cryptographic methods, and to offer a way to compare the myriad cryptographic schemes in use today. The second is to provide some real examples of cryptography in use today.

I would like to say at the outset that this paper is very focused on terms, concepts, and schemes in *current* use and is not a treatise of the whole field. No mention is made here about pre-computerized crypto schemes, the difference between a substitution and transposition cipher, cryptanalysis, or other history. Interested readers should check out some of the books in the references section below for detailed — and interesting! — background information.

2. THE PURPOSE OF CRYPTOGRAPHY

Cryptography is the science of writing in secret code and is an ancient art; the first documented use of cryptography in writing dates back to circa 1900 B.C. when an Egyptian scribe used non-standard hieroglyphs in an inscription. Some experts argue that cryptography appeared spontaneously sometime after writing was invented, with applications ranging from diplomatic missives to war-time battle plans. It is no surprise, then, that new forms of cryptography came soon after the widespread development of computer communications. In data and telecommunications, cryptography is necessary when communicating over any untrusted medium, which includes just about *any* network, particularly the Internet.

Within the context of any application-to-application communication, there are some specific security requirements, including:

- *Authentication:* The process of proving one's identity. (The primary forms of host-to-host authentication on the Internet today are name-based or address-based, both of which are notoriously weak.)
- *Privacy/confidentiality:* Ensuring that no one can read the message except the intended receiver.
- *Integrity:* Assuring the receiver that the received message has not been altered in any way from the original.
- *Non-repudiation:* A mechanism to prove that the sender really sent this message.

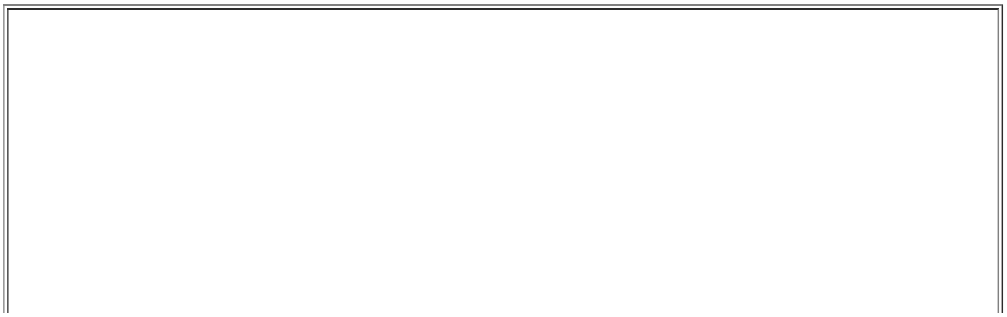
Cryptography, then, not only protects data from theft or alteration, but can also be used for user authentication. There are, in general, three types of cryptographic schemes typically used to accomplish these goals: secret key (or symmetric) cryptography, public-key (or asymmetric) cryptography, and hash functions, each of which is described below. In all cases, the initial unencrypted data is referred to as *plaintext*. It is encrypted into *ciphertext*, which will in turn (usually) be decrypted into usable plaintext.

In many of the descriptions below, two communicating parties will be referred to as Alice and Bob; this is the common nomenclature in the crypto field and literature to make it easier to identify the communicating parties. If there is a third or fourth party to the communication, they will be referred to as Carol and Dave. Mallory is a malicious party, Eve is an eavesdropper, and Trent is a trusted third party.

3. TYPES OF CRYPTOGRAPHIC ALGORITHMS

There are several ways of classifying cryptographic algorithms. For purposes of this paper, they will be categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use. The three types of algorithms that will be discussed are (Figure 1):

- Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption
- Public Key Cryptography (PKC): Uses one key for encryption and another for decryption
- Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information



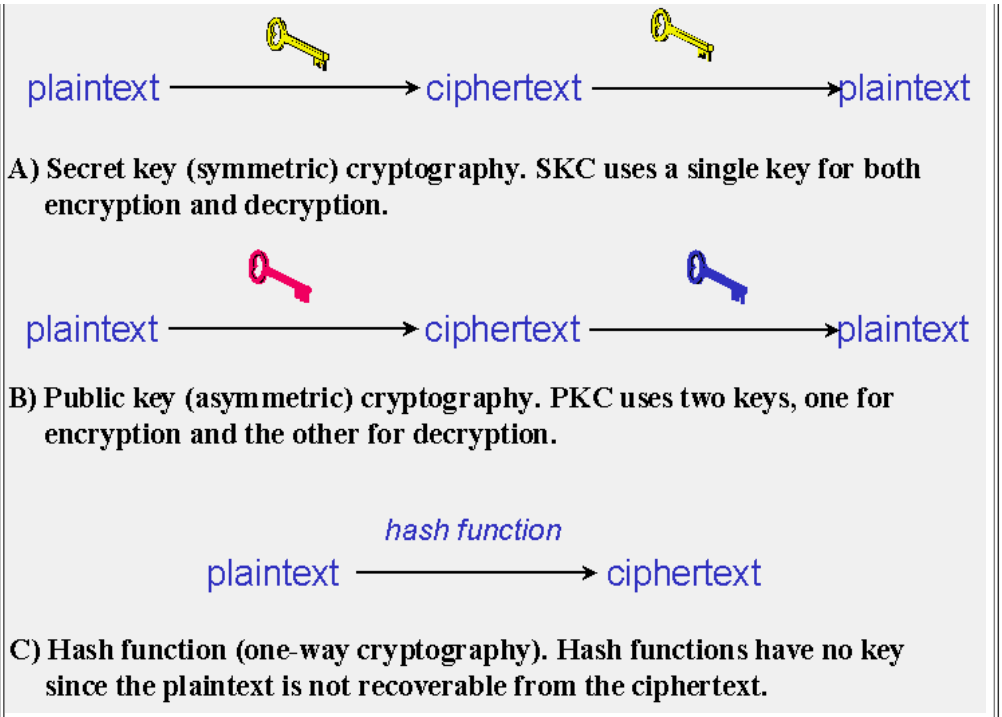


FIGURE 1: Three types of cryptography: secret-key, public key, and hash function.

3.1. Secret Key Cryptography

With *secret key cryptography*, a single key is used for both encryption and decryption. As shown in Figure 1A, the sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key (or ruleset) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called *symmetric encryption*.

With this form of cryptography, it is obvious that the key must be known to both the sender and the receiver; that, in fact, is the secret. The biggest difficulty with this approach, of course, is the distribution of the key.

Secret key cryptography schemes are generally categorized as being either *stream ciphers* or *block ciphers*. Stream ciphers operate on a single bit (byte or computer word) at a time and implement some form of feedback mechanism so that the key is constantly changing. A block cipher is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will always encrypt to the same ciphertext when using the same key in a block cipher whereas the same plaintext will encrypt to different ciphertext in a stream cipher.

Stream ciphers come in several flavors but two are worth mentioning here. *Self-synchronizing stream ciphers* calculate each bit in the keystream as a function of the previous *n* bits in the keystream. It is termed "self-synchronizing" because the decryption process can stay synchronized with the encryption process merely by knowing how far into the *n*-bit keystream it is. One problem is error propagation; a garbled bit in transmission will result in *n* garbled bits at the receiving side. *Synchronous stream ciphers* generate the keystream in a fashion independent of the message stream but by using the same keystream generation function at sender and receiver. While stream ciphers do not propagate transmission errors, they are, by their nature, periodic so that the keystream will eventually repeat.

Block ciphers can operate in one of several modes; the following four are the most important:

- *Electronic Codebook (ECB) mode* is the simplest, most obvious application: the secret key is used to encrypt the plaintext block to form a ciphertext block. Two identical plaintext blocks, then, will always generate the same ciphertext block. Although this is the most common mode of block ciphers, it is susceptible to a variety of brute-force attacks.
- *Cipher Block Chaining (CBC) mode* adds a feedback mechanism to the encryption scheme. In CBC, the plaintext is exclusively-ORed (XORed) with the previous ciphertext block prior to encryption. In this mode, two identical blocks of plaintext never encrypt to the same ciphertext.
- *Cipher Feedback (CFB) mode* is a block cipher implementation as a self-synchronizing stream cipher. CFB mode allows data to be encrypted in units smaller than the block size, which might be useful in some applications such as encrypting interactive terminal input. If we were using 1-byte CFB mode, for example, each incoming character is placed into a shift register the same size as the block, encrypted, and the block transmitted. At the receiving side, the

ciphertext is decrypted and the extra bits in the block (i.e., everything above and beyond the one byte) are discarded.
- *Output Feedback (OFB) mode* is a block cipher implementation conceptually similar to a synchronous stream cipher. OFB prevents the same plaintext block from generating the same ciphertext block by using an internal feedback mechanism that is independent of both the plaintext and ciphertext bitstreams.

A nice overview of these different modes can be found at CRYPTO-IT.

Secret key cryptography algorithms that are in use today include:

- *Data Encryption Standard (DES):* The most common SKC scheme used today, DES was designed by IBM in the 1970s and adopted by the National Bureau of Standards (NBS) [now the National Institute for Standards and Technology (NIST)] in 1977 for commercial and unclassified government applications. DES is a block-cipher employing a 56-bit key that operates on 64-bit blocks. DES has a complex set of rules and transformations that were designed specifically to yield fast hardware implementations and slow software implementations, although this latter point is becoming less significant today since the speed of computer processors is several orders of magnitude faster today than twenty years ago. IBM also proposed a 112-bit key for DES, which was rejected at the time by the government; the use of 112-bit keys was considered in the 1990s, however, conversion was never seriously considered.

  DES is defined in American National Standard X3.92 and three Federal Information Processing Standards (FIPS):

  - FIPS 46-3: DES
  - FIPS 74: Guidelines for Implementing and Using the NBS Data Encryption Standard
  - FIPS 81: DES Modes of Operation

  Information about vulnerabilities of DES can be obtained from the Electronic Frontier Foundation.

  Two important variants that strengthen DES are:

  - *Triple-DES (3DES):* A variant of DES that employs up to three 56-bit keys and makes three encryption/decryption passes over the block; 3DES is also described in FIPS 46-3 and is the recommended replacement to DES.

  - *DESX:* A variant devised by Ron Rivest. By combining 64 additional key bits to the plaintext prior to encryption, effectively increases the keylength to 120 bits.

  More detail about DES, 3DES, and DESX can be found below in Section 5.4.

- *Advanced Encryption Standard (AES):* In 1997, NIST initiated a very public, 4-1/2 year process to develop a new secure cryptosystem for U.S. government applications. The result, the Advanced Encryption Standard, became the official successor to DES in December 2001. AES uses an SKC scheme called Rijndael, a block cipher designed by Belgian cryptographers Joan Daemen and Vincent Rijmen. The algorithm can use a variable block length and key length; the latest specification allowed any combination of keys lengths of 128, 192, or 256 bits and blocks of length 128, 192, or 256 bits. NIST initially selected Rijndael in October 2000 and formal adoption as the AES standard came in December 2001. FIPS PUB 197 describes a 128-bit block cipher employing a 128-, 192-, or 256-bit key. The AES process and Rijndael algorithm are described in more detail below in Section 5.9.

- *CAST-128/256:* CAST-128, described in Request for Comments (RFC) 2144, is a DES-like substitution-permutation crypto algorithm, employing a 128-bit key operating on a 64-bit block. CAST-256 (RFC 2612) is an extension of CAST-128, using a 128-bit block size and a variable length (128, 160, 192, 224, or 256 bit) key. CAST is named for its developers, Carlisle Adams and Stafford Tavares and is available internationally. CAST-256 was one of the Round 1 algorithms in the AES process.

- *International Data Encryption Algorithm (IDEA):* Secret-key cryptosystem written by Xuejia Lai and James Massey, in 1992 and patented by Ascom; a 64-bit SKC block cipher using a 128-bit key. Also available internationally.

- *Rivest Ciphers (*aka *Ron's Code):* Named for Ron Rivest, a series of SKC algorithms.

  - *RC1:* Designed on paper but never implemented.

  - *RC2:* A 64-bit block cipher using variable-sized keys designed to replace DES. It's code has not been made public although many companies have licensed RC2 for use in their products. Described in RFC 2268.

  - *RC3:* Found to be breakable during development.

  - *RC4:* A stream cipher using variable-sized keys; it is widely used in commercial cryptography products. An update to RC4, called Spritz, was designed by Rivest and Jacob Schuldt. More detail about RC4 (and a little about Spritz) can be found below in Section 5.13.

  - *RC5:* A block-cipher supporting a variety of block sizes (32, 64, or 128 bits), key sizes, and number of

encryption passes over the data. Described in RFC 2040.

  - *RC6:* A 128-bit block cipher based upon, and an improvement over, RC5; RC6 was one of the AES Round 2 algorithms.

- *Blowfish:* A symmetric 64-bit block cipher invented by Bruce Schneier; optimized for 32-bit processors with large data caches, it is significantly faster than DES on a Pentium/PowerPC-class machine. Key lengths can vary from 32 to 448 bits in length. Blowfish, available freely and intended as a substitute for DES or IDEA, is in use in a large number of products.

- *Twofish:* A 128-bit block cipher using 128-, 192-, or 256-bit keys. Designed to be highly secure and highly flexible, well-suited for large microprocessors, 8-bit smart card microprocessors, and dedicated hardware. Designed by a team led by Bruce Schneier and was one of the Round 2 algorithms in the AES process.

- *Camellia:* A secret-key, block-cipher crypto algorithm developed jointly by Nippon Telegraph and Telephone (NTT) Corp. and Mitsubishi Electric Corporation (MEC) in 2000. Camellia has some characteristics in common with AES: a 128-bit block size, support for 128-, 192-, and 256-bit key lengths, and suitability for both software and hardware implementations on common 32-bit processors as well as 8-bit processors (e.g., smart cards, cryptographic hardware, and embedded systems). Also described in RFC 3713. Camellia's application in IPsec is described in RFC 4312 and application in OpenPGP in RFC 5581.

- *MISTY1:* Developed at Mitsubishi Electric Corp., a block cipher using a 128-bit key and 64-bit blocks, and a variable number of rounds. Designed for hardware and software implementations, and is resistant to differential and linear cryptanalysis. Described in RFC 2994.

- *Secure and Fast Encryption Routine (SAFER):* Secret-key crypto scheme designed for implementation in software. Versions have been defined for 40-, 64-, and 128-bit keys.

- *KASUMI:* A block cipher using a 128-bit key that is part of the Third-Generation Partnership Project (3gpp), formerly known as the Universal Mobile Telecommunications System (UMTS). KASUMI is the intended confidentiality and integrity algorithm for both message content and signaling data for emerging mobile communications systems.

- *SEED:* A block cipher using 128-bit blocks and 128-bit keys. Developed by the Korea Information Security Agency (KISA) and adopted as a national standard encryption algorithm in South Korea. Also described in RFC 4269.

- *ARIA:* A 128-bit block cipher employing 128-, 192-, and 256-bit keys. Developed by large group of researchers from academic institutions, research institutes, and federal agencies in South Korea in 2003, and subsequently named a national standard. Described in RFC 5794.

- *CLEFIA:* Described in RFC 6114, CLEFIA is a 128-bit block cipher employing key lengths of 128, 192, and 256 bits (which is compatible with AES). The CLEFIA algorithm was first published in 2007 by Sony Corporation. CLEFIA is one of the new-generation lightweight blockcipher algorithms designed after AES, offering high performance in software and hardware as well as a lightweight implementation in hardware.

- *SMS4:* SMS4 is a 128-bit block cipher using 128-bit keys and 32 rounds to process a block. Declassified in 2006, SMS4 is used in the Chinese National Standard for Wireless Local Area Network (LAN) Authentication and Privacy Infrastructure (WAPI). SMS4 had been a proposed cipher for the Institute of Electrical and Electronics Engineers (IEEE) 802.11i standard on security mechanisms for wireless LANs, but has yet to be accepted by the IEEE or International Organization for Standardization (ISO). SMS4 is described in *SMS4 Encryption Algorithm for Wireless Networks* (translated and typeset by Whitfield Diffie and George Ledin, 2008) or in the original Chinese.

- *Skipjack:* SKC scheme proposed for Capstone. Although the details of the algorithm were never made public, Skipjack was a block cipher using an 80-bit key and 32 iteration cycles per 64-bit block.

- *GSM (Global System for Mobile Communications, originally Groupe Spécial Mobile) encryption:* GSM mobile phone systems use several stream ciphers for over-the-air communication privacy. A5/1 was developed in 1987 for use in Europe and the U.S. A5/2, developed in 1989, is a weaker algorithm and intended for use outside of Europe and the U.S. Significant flaws were found in both ciphers after the "secret" specifications were leaked in 1994, however, and A5/2 has been withdrawn from use. The newest version, A5/3, employs the KASUMI block cipher. **NOTE:** Unfortunately, although A5/1 has been repeatedly "broken" (e.g., see "Secret code protecting cellphone calls set loose" [2009] and "Cellphone snooping now easier and cheaper than ever" [2011]), this encryption scheme remains in widespread use, even in 3G and 4G mobile phone networks. Use of this scheme is reportedly one of the reasons that the National Security Agency (NSA) can easily decode voice and data calls over mobile phone networks.

- *GPRS (General Packet Radio Service) encryption:* GSM mobile phone systems use GPRS for data applications, and GPRS uses a number of encryption methods, offering different levels of data protection. GEA/0 offers no encryption at all. GEA/1 and GEA/2 are proprietary stream ciphers, employing a 64-bit key and a 96-bit or 128-bit state,

respectively. GEA/1 and GEA/2 are most widely used by network service providers today although both have been reportedly broken. GEA/3 is a 128-bit block cipher employing a 64-bit key that is used by some carriers; GEA/4 is a 128-bit clock cipher with a 128-bit key, but is not yet deployed.

- *KCipher-2:* Described in RFC 7008, KCipher-2 is a stream cipher with a 128-bit key and a 128-bit initialization vector. Using simple arithmetic operations, the algorithms offers fast encryption and decryption by use of efficient implementations. KCipher-2 has been used for industrial applications, especially for mobile health monitoring and diagnostic services in Japan.

3.2. Public-Key Cryptography

Public-key cryptography has been said to be the most significant new development in cryptography in the last 300-400 years. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

PKC depends upon the existence of so-called *one-way functions*, or mathematical functions that are easy to compute whereas their inverse function is relatively difficult to compute. Let me give you two simple examples:

1. *Multiplication vs. factorization:* Suppose I tell you that I have two prime numbers, 3 and 7, and that I want to calculate the product; it should take almost no time to calculate that value, which is 21. Now suppose, instead, that I tell you that I have a number, 21, and I need you tell me which pair of prime numbers I multiplied together to obtain that number. You will eventually come up with the solution but whereas calculating the product took milliseconds, factoring will take longer. The problem becomes much harder if I start with primes that have 400 digits or so, because the product will have ~800 digits.
2. *Exponentiation vs. logarithms:* Suppose I tell you that I want to take the number 3 to the 6th power; again, it is relatively easy to calculate $3^6 = 729$. But if I tell you that I have the number 729 and want you to tell me the two integers that I used, *x* and *y* so that $\log_x 729 = y$, it will take you longer to find the two values.

While the examples above are trivial, they do represent two of the functional pairs that are used with PKC; namely, the ease of multiplication and exponentiation versus the relative difficulty of factoring and calculating logarithms, respectively. The mathematical "trick" in PKC is to find a *trap door* in the one-way function so that the inverse calculation becomes easy given knowledge of some item of information.

Generic PKC employs two keys that are mathematically related although knowledge of one key does not allow someone to easily determine the other key. One key is used to encrypt the plaintext and the other key is used to decrypt the ciphertext. The important point here is that it **does not matter which key is applied first**, but that both keys are required for the process to work (Figure 1B). Because a pair of keys are required, this approach is also called *asymmetric cryptography*.

In PKC, one of the keys is designated the *public key* and may be advertised as widely as the owner wants. The other key is designated the *private key* and is never revealed to another party. It is straight forward to send messages under this scheme. Suppose Alice wants to send Bob a message. Alice encrypts some information using Bob's public key; Bob decrypts the ciphertext using his private key. This method could be also used to prove who sent a message; Alice, for example, could encrypt some plaintext with her private key; when Bob decrypts using Alice's public key, he knows that Alice sent the message and Alice cannot deny having sent the message (*non-repudiation*).

Public-key cryptography algorithms that are in use today for key exchange or digital signatures include:

- *RSA:* The first, and still most common, PKC implementation, named for the three MIT mathematicians who developed it — Ronald Rivest, Adi Shamir, and Leonard Adleman. RSA today is used in hundreds of software products and can be used for key exchange, digital signatures, or encryption of small blocks of data. RSA uses a variable size encryption block and a variable size key. The key-pair is derived from a very large number, *n*, that is the product of two prime numbers chosen according to special rules; these primes may be 100 or more digits in length each, yielding an *n* with roughly twice as many digits as the prime factors. The public key information includes *n* and a derivative of one of the factors of *n*; an attacker cannot determine the prime factors of *n* (and, therefore, the private key) from this information alone and that is what makes the RSA algorithm so secure. (Some descriptions of PKC erroneously state that RSA's safety is due to the difficulty in *factoring* large prime numbers. In fact, large prime numbers, like small prime numbers, only have two factors!) The ability for computers to factor large numbers, and therefore attack schemes such as RSA, is rapidly improving and systems today can find the prime factors of numbers with more than 200 digits. Nevertheless, if a large number is created from two prime factors that are roughly the same size, there is no known factorization algorithm that will solve the problem in a reasonable amount of time; a 2005 test to factor a 200-digit number took 1.5 years and over 50 years of compute time (see the Wikipedia article on integer factorization.) Regardless, one presumed protection of RSA is that users can easily increase the key size to always stay ahead of the computer processing curve. As an aside, the patent for RSA expired in September 2000 which does not appear to have affected RSA's popularity one way or the other. A detailed example of RSA is presented below in Section 5.3.

- *Diffie-Hellman:* After the RSA algorithm was published, Diffie and Hellman came up with their own algorithm. D-H is used for secret-key key exchange only, and not for authentication or digital signatures. More detail about Diffie-Hellman can be found below in Section 5.2.

- *Digital Signature Algorithm (DSA):* The algorithm specified in NIST's Digital Signature Standard (DSS), provides digital signature capability for the authentication of messages.

- *ElGamal:* Designed by Taher Elgamal, a PKC system similar to Diffie-Hellman and used for key exchange.

- *Elliptic Curve Cryptography (ECC):* A PKC algorithm based upon elliptic curves. ECC can offer levels of security with small keys comparable to RSA and other PKC methods. It was designed for devices with limited compute power and/or memory, such as smartcards and PDAs. More detail about ECC can be found below in Section 5.8. Other references include "The Importance of ECC" Web page and the "Online Elliptic Curve Cryptography Tutorial", both from Certicom. See also RFC 6090 for a review of fundamental ECC algorithms and *The Elliptic Curve Digital Signature Algorithm (ECDSA)* for details about the use of ECC for digital signatures.

- *Public-Key Cryptography Standards (PKCS):* A set of interoperable standards and guidelines for public-key cryptography, designed by RSA Data Security Inc.

    - PKCS #1: RSA Cryptography Standard (Also RFC 3447)
    - PKCS #2: *Incorporated into PKCS #1.*
    - PKCS #3: Diffie-Hellman Key-Agreement Standard
    - PKCS #4: *Incorporated into PKCS #1.*
    - PKCS #5: Password-Based Cryptography Standard (PKCS #5 V2.0 is also RFC 2898)
    - PKCS #6: Extended-Certificate Syntax Standard (being phased out in favor of X.509v3)
    - PKCS #7: Cryptographic Message Syntax Standard (Also RFC 2315)
    - PKCS #8: Private-Key Information Syntax Standard (Also RFC 5208)
    - PKCS #9: Selected Attribute Types (Also RFC 2985)
    - PKCS #10: Certification Request Syntax Standard (Also RFC 2986)
    - PKCS #11: Cryptographic Token Interface Standard
    - PKCS #12: Personal Information Exchange Syntax Standard (Also RFC 7292)
    - PKCS #13: Elliptic Curve Cryptography Standard
    - PKCS #14: *Pseudorandom Number Generation Standard is no longer available*
    - PKCS #15: Cryptographic Token Information Format Standard

- *Cramer-Shoup:* A public-key cryptosystem proposed by R. Cramer and V. Shoup of IBM in 1998.

- *Key Exchange Algorithm (KEA):* A variation on Diffie-Hellman; proposed as the key exchange method for Capstone.

- *LUC:* A public-key cryptosystem designed by P.J. Smith and based on Lucas sequences. Can be used for encryption and signatures, using integer factoring.

For additional information on PKC algorithms, see "Public-Key Encryption" (Chapter 8) in *Handbook of Applied Cryptography*, by A. Menezes, P. van Oorschot, and S. Vanstone (CRC Press, 1996).

A digression: Who invented PKC? I tried to be careful in the first paragraph of this section to state that Diffie and Hellman "first described publicly" a PKC scheme. Although I have categorized PKC as a two-key system, that has been merely for convenience; the real criteria for a PKC scheme is that it allows two parties to exchange a secret even though the communication with the shared secret might be overheard. There seems to be no question that Diffie and Hellman were first to publish; their method is described in the classic paper, "New Directions in Cryptography," published in the November 1976 issue of *IEEE Transactions on Information Theory* (*IT-22*(6), 644-654). As shown below, Diffie-Hellman uses the idea that finding logarithms is relatively harder than performing exponentiation. And, indeed, it is the precursor to modern PKC which does employ two keys. Rivest, Shamir, and Adleman described an implementation that extended this idea in their paper "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," published in the February 1978 issue of the *Communications of the ACM (CACM)* (*21*92), 120-126). Their method, of course, is based upon the relative ease of finding the product of two large prime numbers compared to finding the prime factors of a large number.

Some sources, though, credit Ralph Merkle with first describing a system that allows two parties to share a secret although it was not a two-key system, per se. A *Merkle Puzzle* works where Alice creates a large number of encrypted keys, sends them all to Bob so that Bob chooses one at random and then lets Alice know which he has selected. An eavesdropper (Eve) will see all of the keys but can't learn which key Bob has selected (because he has encrypted the response with the chosen key). In this case, Eve's effort to break in is the square of the effort of Bob to choose a key. While this difference may be small it is often sufficient. Merkle apparently took a computer science course at UC Berkeley in 1974 and described his method, but had difficulty making people understand it; frustrated, he dropped the course. Meanwhile, he submitted the paper "Secure Communication Over Insecure Channels" which was published in the *CACM* in April 1978; Rivest et al.'s paper even makes

reference to it. Merkle's method certainly wasn't published first, but did he have the idea first?

An interesting question, maybe, but who really knows? For some time, it was a quiet secret that a team at the UK's Government Communications Headquarters (GCHQ) had first developed PKC in the early 1970s. Because of the nature of the work, GCHQ kept the original memos classified. In 1997, however, the GCHQ changed their posture when they realized that there was nothing to gain by continued silence. Documents show that a GCHQ mathematician named James Ellis started research into the key distribution problem in 1969 and that by 1975, Ellis, Clifford Cocks, and Malcolm Williamson had worked out all of the fundamental details of PKC, yet couldn't talk about their work. (They were, of course, barred from challenging the RSA patent!) After more than 20 years, Ellis, Cocks, and Williamson have begun to get their due credit.

And the National Security Agency (NSA) claims to have knowledge of this type of algorithm as early as 1966 but there is no supporting documentation... yet. So this really was a digression...

3.3. Hash Functions

Hash functions, also called *message digests* and *one-way encryption*, are algorithms that, in some sense, use no key (Figure 1C). Instead, a fixed-length hash value is computed based upon the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered. Hash algorithms are typically used to provide a *digital fingerprint* of a file's contents, often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating systems to encrypt passwords. Hash functions, then, provide a measure of the integrity of a file.

Hash algorithms that are in common use today include:

- *Message Digest (MD) algorithms:* A series of byte-oriented algorithms that produce a 128-bit hash value from an arbitrary-length message.
    - *MD2 (RFC 1319):* Designed for systems with limited memory, such as smart cards. (MD2 has been relegated to historical status, per RFC 6149.)
    - *MD4 (RFC 1320):* Developed by Rivest, similar to MD2 but designed specifically for fast processing in software. (MD4 has been relegated to historical status, per RFC 6150.)
    - *MD5 (RFC 1321):* Also developed by Rivest after potential weaknesses were reported in MD4; this scheme is similar to MD4 but is slower because more manipulation is made to the original data. MD5 has been implemented in a large number of products although several weaknesses in the algorithm were demonstrated by German cryptographer Hans Dobbertin in 1996 ("Cryptanalysis of MD5 Compress").
- *Secure Hash Algorithm (SHA):* Algorithm for NIST's Secure Hash Standard (SHS).
    - SHA-1 produces a 160-bit hash value and was originally published as FIPS PUB 180-1 and RFC 3174.
    - SHA-2, originally described in FIPS PUB 180-2 and eventually replaced by FIPS PUB 180-3 and FIPS PUB 180-4), comprises five algorithms in the SHS: SHA-1 plus SHA-224, SHA-256, SHA-384, and SHA-512 which can produce hash values that are 224, 256, 384, or 512 bits in length, respectively. SHA-2 recommends use of SHA-1, SHA-224, and SHA-256 for messages less than 2^{64} bits in length, and employs a 512 bit block size; SHA-384 and SHA-512 are recommended for messages less than 2^{128} bits in length, and employs a 1,024 bit block size. FIPS PUB 180-4 also introduces the concept of a truncated hash in SHA-512/*t*, a generic name referring to a hash value based upon the SHA-512 algorithm that has been truncated to *t* bits; SHA-512/224 and SHA-512/256 are specifically described. SHA-224, -256, -384, and -512 are also described in RFC 4634.
    - SHA-3 is a proposed new SHS algorithm. Although there have not been any successful attacks on SHA-2, NIST decided that having an alternative to SHA-2 using a different algorithm would be prudent. In 2007, they launched a SHA-3 Competition to find that alternative. In 2012, NIST announced that after reviewing 64 submissions, the winner was Keccak (pronounced "catch-ack"). Based upon a sponge function — which is different from the algorithm used for SHA-1 and SHA-2 — SHA-3 will employ the same hash lengths as SHA-2. SHA-3 should be published by NIST by the middle of 2014.
- *RIPEMD:* A series of message digests that initially came from the RIPE (RACE Integrity Primitives Evaluation) project. RIPEMD-160 was designed by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel, and optimized for 32-bit processors to replace the then-current 128-bit hash functions. Other versions include RIPEMD-256, RIPEMD-320, and RIPEMD-128.
- *HAVAL (HAsh of VAriable Length):* Designed by Y. Zheng, J. Pieprzyk and J. Seberry, a hash algorithm with many levels of security. HAVAL can create hash values that are 128, 160, 192, 224, or 256 bits in length.
- *Whirlpool:* A relatively new hash function, designed by V. Rijmen and P.S.L.M. Barreto. Whirlpool operates on

messages less than 2^{256} bits in length, and produces a message digest of 512 bits. The design of this has function is very different than that of MD5 and SHA-1, making it immune to the same attacks as on those hashes (see below).

- *Tiger*: Designed by Ross Anderson and Eli Biham, Tiger is designed to be secure, run efficiently on 64-bit processors, and easily replace MD4, MD5, SHA and SHA-1 in other applications. Tiger/192 produces a 192-bit output and is compatible with 64-bit architectures; Tiger/128 and Tiger/160 produce a hash of length 128 and 160 bits, respectively, to provide compatibility with the other hash functions mentioned above.

(Readers might be interested in HashCalc, a Windows-based program that calculates hash values using a dozen algorithms, including MD5, SHA-1 and several variants, RIPEMD-160, and Tiger. Command line utilities that calculate hash values include sha_verify by Dan Mares [Windows; supports MD5, SHA-1, SHA-2] and md5deep [cross-platform; supports MD5, SHA-1, SHA-256, Tiger, and Whirlpool].)

Hash functions are sometimes misunderstood and some sources claim that no two files can have the same hash value. This is, in fact, not correct. Consider a hash function that provides a 128-bit hash value. There are, obviously, 2^{128} possible hash values. But there are an infinite number of possible files and $\infty >> 2^{128}$. Therefore, there have to be multiple files — in fact, there have to be an infinite number of files! — that can have the same 128-bit hash value.

The difficulty is *finding* two files with the same hash! What is, indeed, very hard to do is to try to create a file that has a given hash value so as to force a hash value collision — which is the reason that hash functions are used extensively for information security and computer forensics applications. Alas, researchers in 2004 found that *practical* collision attacks could be launched on MD5, SHA-1, and other hash algorithms. Readers interested in this problem should read the following:

- AccessData. (2006, April). *MD5 Collisions: The Effect on Computer Forensics*. AccessData White Paper.
- Burr, W. (2006, March/April). Cryptographic hash standards: Where do we go from here? *IEEE Security & Privacy*, *4*(2), 88-91.
- Dwyer, D. (2009, June 3). SHA-1 Collision Attacks Now 2^{52}. *SecureWorks Research blog*.
- Gutman, P., Naccache, D., & Palmer, C.C. (2005, May/June). When hashes collide. *IEEE Security & Privacy*, *3*(3), 68-71.
- Klima, V. (March 2005). Finding MD5 Collisions - a Toy For a Notebook.
- Lee, R. (2009, January 7). Law Is Not A Science: Admissibility of Computer Evidence and MD5 Hashes. *SANS Computer Forensics blog*.
- Thompson, E. (2005, February). MD5 collisions and the impact on computer forensics. *Digital Investigation*, *2*(1), 36-40.
- Wang, X., Feng, D., Lai, X., & Yu, H. (2004, August). Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD.
- Wang, X., Yin, Y.L., & Yu, H. (2005, February 13). Collision Search Attacks on SHA1.

Readers are also referred to the Eindhoven University of Technology HashClash Project Web site. An excellent overview of the situation with hash collisions (circa 2005) can be found in RFC 4270 (by P. Hoffman and B. Schneier, November 2005). And for additional information on hash functions, see David Hopwood's MessageDigest Algorithms page. Finally, for an interesting twist on this discussion, read about the *Nostradamus* attack reported at Predicting the winner of the 2008 US Presidential Elections using a Sony PlayStation 3 (by M. Stevens, A.K. Lenstra, and B. de Weger, November 2007).

At this time, there is no successor to MD5 and SHA-1 that can be put into use quickly; there are so many products using these hash functions that it could take many years to flush out all use of 128- and 160-bit hashes. Meanwhile, NIST announced in 2007 their SHA-3 Competition to find the next-generation secure hashing method. A list of submissions can be found at The SHA-3 Zoo. In 2012, NIST declared Keccak as the SHA-3 winner. Keccak is a family of so-called sponge functions, and the NIST version can support hash output sizes of 256 and 512 bits.

Certain extensions of hash functions are used for a variety of information security and digital forensics applications, such as:

- *Hash libraries* are sets of hash values corresponding to known files. A hash library of known good files, for example, might be a set of files known to be a part of an operating system, while a hash library of known bad files might be of a set of known child pornographic images.
- *Rolling hashes* refer to a set of hash values that are computed based upon a fixed-length "sliding window" through the input. As an example, a hash value might be computed on bytes 1-10 of a file, then on bytes 2-11, 3-12, 4-13, etc.
- *Fuzzy hashes* are an area of intense research and represent hash values that represent two inputs that are similar. Fuzzy hashes are used to detect documents, images, or other files that are close to each other with respect to content. See "Fuzzy Hashing" (PDF | PPT) by Jesse Kornblum for a good treatment of this topic.

3.4. Why Three Encryption Techniques?

So, why are there so many different types of cryptographic schemes? Why can't we do everything we need with just one?

The answer is that each scheme is optimized for some specific application(s). Hash functions, for example, are well-suited

for ensuring data integrity because any change made to the contents of a message will result in the receiver calculating a different hash value than the one placed in the transmission by the sender. Since it is highly unlikely that two different messages will yield the same hash value, data integrity is ensured to a high degree of confidence.

Secret key cryptography, on the other hand, is ideally suited to encrypting messages, thus providing privacy and confidentiality. The sender can generate a *session key* on a per-message basis to encrypt the message; the receiver, of course, needs the same session key to decrypt the message.

Key exchange, of course, is a key application of public-key cryptography (no pun intended). Asymmetric schemes can also be used for non-repudiation and user authentication; if the receiver can obtain the session key encrypted with the sender's private key, then only this sender could have sent the message. Public-key cryptography could, theoretically, also be used to encrypt messages although this is rarely done because secret-key cryptography operates about 1000 times faster than public-key cryptography.

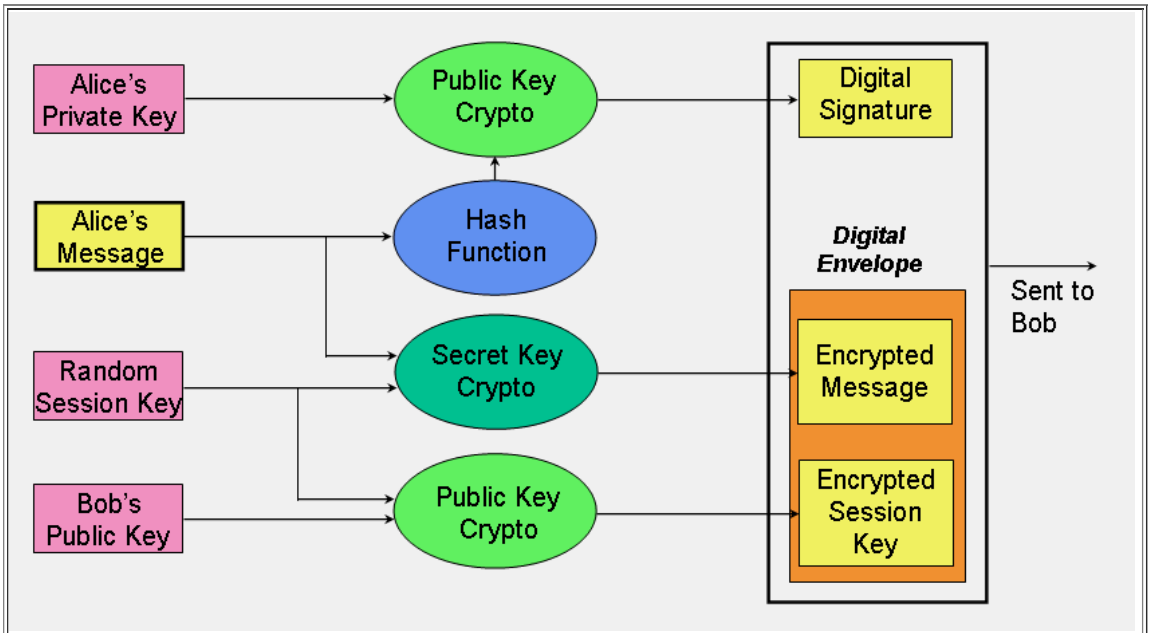


FIGURE 2: Sample application of the three cryptographic techniques for secure communication.

Figure 2 puts all of this together and shows how a *hybrid cryptographic* scheme combines all of these functions to form a secure transmission comprising *digital signature* and *digital envelope*. In this example, the sender of the message is Alice and the receiver is Bob.

A digital envelope comprises an encrypted message and an encrypted session key. Alice uses secret key cryptography to encrypt her message using the *session key*, which she generates at random with each session. Alice then encrypts the session key using Bob's public key. The encrypted message and encrypted session key together form the digital envelope. Upon receipt, Bob recovers the session secret key using his private key and then decrypts the encrypted message.

The digital signature is formed in two steps. First, Alice computes the hash value of her message; next, she encrypts the hash value with her private key. Upon receipt of the digital signature, Bob recovers the hash value calculated by Alice by decrypting the digital signature with Alice's public key. Bob can then apply the hash function to Alice's original message, which he has already decrypted (see previous paragraph). If the resultant hash value is not the same as the value supplied by Alice, then Bob knows that the message has been altered; if the hash values are the same, Bob should believe that the message he received is identical to the one that Alice sent.

This scheme also provides nonrepudiation since it proves that Alice sent the message; if the hash value recovered by Bob using Alice's public key proves that the message has not been altered, then only Alice could have created the digital signature. Bob also has proof that he is the intended receiver; if he can correctly decrypt the message, then he must have correctly decrypted the session key meaning that his is the correct private key.

This diagram purposely suggests a cryptosystem where the session key is used for just a single session. Even if this session key is somehow broken, only this session will be compromised; the session key for the next session is in no way based upon the key for this session, just as this session's key is not dependent on the key from the previous session. This is known as Perfect Forward Secrecy; you might lose one session key due to a compromise but you won't lose all of them. (This was an issue in the 2014 OpenSSL vulnerability known as Heartbleed.)

3.5. The Significance of Key Length

In a 1998 article in the industry literature, a writer made the claim that 56-bit keys did not provide as adequate protection for DES at that time as they did in 1975 because computers were 1000 times faster in 1998 than in 1975. Therefore, the writer went on, we needed 56,000-bit keys in 1998 instead of 56-bit keys to provide adequate protection. The conclusion was then drawn that because 56,000-bit keys are infeasible (*true*), we should accept the fact that we have to live with weak cryptography (*false!*). The major error here is that the writer did not take into account that the number of possible key values double whenever a single bit is added to the key length; thus, a 57-bit key has twice as many values as a 56-bit key (because 2^{57} is two times 2^{56}). In fact, a 66-bit key would have 1024 times more values than a 56-bit key.

But this does bring up the issue, what is the precise significance of key length as it affects the level of protection?

In cryptography, size does matter. The larger the key, the harder it is to crack a block of encrypted data. The reason that large keys offer more protection is almost obvious; computers have made it easier to attack ciphertext by using brute force methods rather than by attacking the mathematics (which are generally well-known anyway). With a brute force attack, the attacker merely generates every possible key and applies it to the ciphertext. Any resulting plaintext that makes sense offers a candidate for a legitimate key. This was the basis, of course, of the EFF's attack on DES.

Until the mid-1990s or so, brute force attacks were beyond the capabilities of computers that were within the budget of the attacker community. By that time, however, significant compute power was typically available and accessible. General-purpose computers such as PCs were already being used for brute force attacks. For serious attackers with money to spend, such as some large companies or governments, Field Programmable Gate Array (FPGA) or Application-Specific Integrated Circuits (ASIC) technology offered the ability to build specialized chips that could provide even faster and cheaper solutions than a PC. (As an example, the AT&T Optimized Reconfigurable Cell Array (ORCA) FPGA chip cost about $200 and could test 30 million DES keys per second, while a $10 ASIC chip could test 200 million DES keys per second; compare that to a PC which might be able to test 40,000 keys per second.) Distributed attacks, harnessing the power of between tens and tens of thousands of powerful CPUs, are now commonly employed to try to brute-force crypto keys.

The table below — from a 1995 article discussing both why exporting 40-bit keys was, in essence, no crypto at all *and* why DES' days were numbered — shows what DES key sizes were needed to protect data from attackers with different time and financial resources. This information was not merely academic; one of the basic tenets of any security system is to have an idea of *what* you are protecting and *from who* are you protecting it! The table clearly shows that a 40-bit key was essentially worthless against even the most unsophisticated attacker. On the other hand, 56-bit keys were fairly strong unless you might be subject to some pretty serious corporate or government espionage. But note that even 56-bit keys were clearly on the decline in their value and that the times in the table were worst cases.

TABLE 1. Minimum Key Lengths for Symmetric Ciphers (1995).

Type of Attacker	Budget	Tool	Time and Cost Per Key Recovered		Key Length Needed For Protection In Late-1995
			40 bits	56 bits	
Pedestrian Hacker	Tiny	Scavenged computer time	1 week	Infeasible	45
	$400	FPGA	5 hours ($0.08)	38 years ($5,000)	50
Small Business	$10,000	FPGA	12 minutes ($0.08)	18 months ($5,000)	55
Corporate Department	$300K	FPGA	24 seconds ($0.08)	19 days ($5,000)	60
		ASIC	0.18 seconds ($0.001)	3 hours ($38)	
Big Company	$10M	FPGA	7 seconds ($0.08)	13 hours ($5,000)	70
		ASIC	0.005 seconds ($0.001)	6 minutes ($38)	
Intelligence Agency	$300M	ASIC	0.0002 seconds ($0.001)	12 seconds ($38)	75

So, how big is big enough? DES, invented in 1975, was still in use at the turn of the century, nearly 25 years later. If we

take that to be a design criteria (i.e., a 20-plus year lifetime) and we believe Moore's Law ("computing power doubles every 18 months"), then a key size extension of 14 bits (i.e., a factor of more than 16,000) should be adequate. The 1975 DES proposal suggested 56-bit keys; by 1995, a 70-bit key would have been required to offer equal protection and an 85-bit key necessary by 2015.

A 256- or 512-bit SKC key will probably suffice for some time because that length keeps us ahead of the brute force capabilities of the attackers. Note that while a large key is good, a huge key may not always be better; for example, expanding PKC keys beyond the current 2048- or 4096-bit lengths doesn't add any necessary protection at this time. Weaknesses in cryptosystems are largely based upon key management rather than weak keys.

Much of the discussion above, including the table, is based on the paper "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security" by M. Blaze, W. Diffie, R.L. Rivest, B. Schneier, T. Shimomura, E. Thompson, and M. Wiener.

The most effective large-number factoring methods today use a mathematical Number Field Sieve to find a certain number of relationships and then uses a matrix operation to solve a linear equation to produce the two prime factors. The sieve step actually involves a large number of operations that can be performed in parallel; solving the linear equation, however, requires a supercomputer. Indeed, finding the solution to the RSA-140 challenge in February 1999 — factoring a 140-digit (465-bit) prime number — required 200 computers across the Internet about 4 weeks for the first step and a Cray computer 100 hours and 810 MB of memory to do the second step.

In early 1999, Shamir (of RSA fame) described a new machine that could increase factorization speed by 2-3 orders of magnitude. Although no detailed plans were provided nor is one known to have been built, the concepts of TWINKLE (The Weizmann Institute Key Locating Engine) could result in a specialized piece of hardware that would cost about $5000 and have the processing power of 100-1000 PCs. There still appear to be many engineering details that have to be worked out before such a machine could be built. Furthermore, the hardware improves the sieve step only; the matrix operation is not optimized at all by this design and the complexity of this step grows rapidly with key length, both in terms of processing time and memory requirements. Nevertheless, this plan conceptually puts 512-bit keys within reach of being factored. Although most PKC schemes allow keys that are 1024 bits and longer, Shamir claims that 512-bit RSA keys "protect 95% of today's E-commerce on the Internet." (See Bruce Schneier's Crypto-Gram (May 15, 1999) for more information, as well as the comments from RSA Labs.)

It is also interesting to note that while cryptography is good and strong cryptography is better, long keys may disrupt the nature of the randomness of data files. Shamir and van Someren ("Playing hide and seek with stored keys") have noted that a new generation of viruses can be written that will find files encrypted with long keys, making them easier to find by intruders and, therefore, more prone to attack.

Finally, U.S. government policy has tightly controlled the export of crypto products since World War II. Until the mid-1990s, export outside of North America of cryptographic products using keys greater than 40 bits in length was prohibited, which made those products essentially worthless in the marketplace, particularly for electronic commerce; today, crypto products are widely available on the Internet without restriction. The U.S. Department of Commerce Bureau of Industry and Security maintains an Encryption FAQ web page with more information about the current state of encryption registration.

On a related topic, public key crypto schemes can be used for several purposes, including key exchange, digital signatures, authentication, and more. In those PKC systems used for SKC key exchange, the PKC key lengths are chosen so to be resistant to some selected level of attack. The length of the secret keys exchanged via that system have to have at least the same level of attack resistance. Thus, the three parameters of such a system — system strength, secret key strength, and public key strength — must be matched. This topic is explored in more detail in *Determining Strengths For Public Keys Used For Exchanging Symmetric Keys* (RFC 3766).

4. TRUST MODELS

Secure use of cryptography requires trust. While secret key cryptography can ensure message confidentiality and hash codes can ensure integrity, none of this works without trust. In SKC, Alice and Bob had to share a secret key. PKC solved the secret distribution problem, but how does Alice really know that Bob is who he says he is? Just because Bob has a public and private key, and purports to be "Bob," how does Alice know that a malicious person (Mallory) is not pretending to be Bob?

There are a number of *trust models* employed by various cryptographic schemes. This section will explore three of them:

- The web of trust employed by Pretty Good Privacy (PGP) users, who hold their own set of trusted public keys.
- Kerberos, a secret key distribution scheme using a trusted third party.
- Certificates, which allow a set of trusted third parties to authenticate each other and, by implication, each other's users.

Each of these trust models differs in complexity, general applicability, scope, and scalability.

4.1. PGP Web of Trust

Pretty Good Privacy (described more below in Section 5.5) is a widely used private e-mail scheme based on public key methods. A PGP user maintains a local keyring of all their known and trusted public keys. The user makes their own determination about the trustworthiness of a key using what is called a "web of trust."

If Alice needs Bob's public key, Alice can ask Bob for it in another e-mail or, in many cases, download the public key from an advertised server; this server might a well-known PGP key repository or a site that Bob maintains himself. In fact, Bob's public key might be stored or listed in many places. (The author's public key, for example, can be found at *http://www.garykessler.net/pubkey.html*.) Alice is prepared to believe that Bob's public key, as stored at these locations, is valid.

Suppose Carol claims to hold Bob's public key and offers to give the key to Alice. How does Alice know that Carol's version of Bob's key is valid or if Carol is actually giving Alice a key that will allow Mallory access to messages? The answer is, "It depends." If Alice trusts Carol and Carol says that she thinks that her version of Bob's key is valid, then Alice *may* — at *her* option — trust that key. And trust is not necessarily transitive; if Dave has a copy of Bob's key and Carol trusts Dave, it does not necessarily follow that Alice trusts Dave even if she does trust Carol.

The point here is that who Alice trusts and how she makes that determination is strictly up to Alice. PGP makes no statement and has no protocol about how one user determines whether they trust another user or not. In any case, encryption and signatures based on public keys can only be used when the appropriate public key is on the user's keyring.

4.2. Kerberos

Kerberos is a commonly used authentication scheme on the Internet. Developed by MIT's Project Athena, Kerberos is named for the three-headed dog who, according to Greek mythology, guards the entrance of Hades (rather than the exit, for some reason!).

Kerberos employs a client/server architecture and provides user-to-server authentication rather than host-to-host authentication. In this model, security and authentication will be based on secret key technology where every host on the network has its own secret key. It would clearly be unmanageable if every host had to know the keys of all other hosts so a secure, trusted host somewhere on the network, known as a Key Distribution Center (KDC), knows the keys for all of the hosts (or at least some of the hosts within a portion of the network, called a *realm*). In this way, when a new node is brought online, only the KDC and the new node need to be configured with the node's key; keys can be distributed physically or by some other secure means.

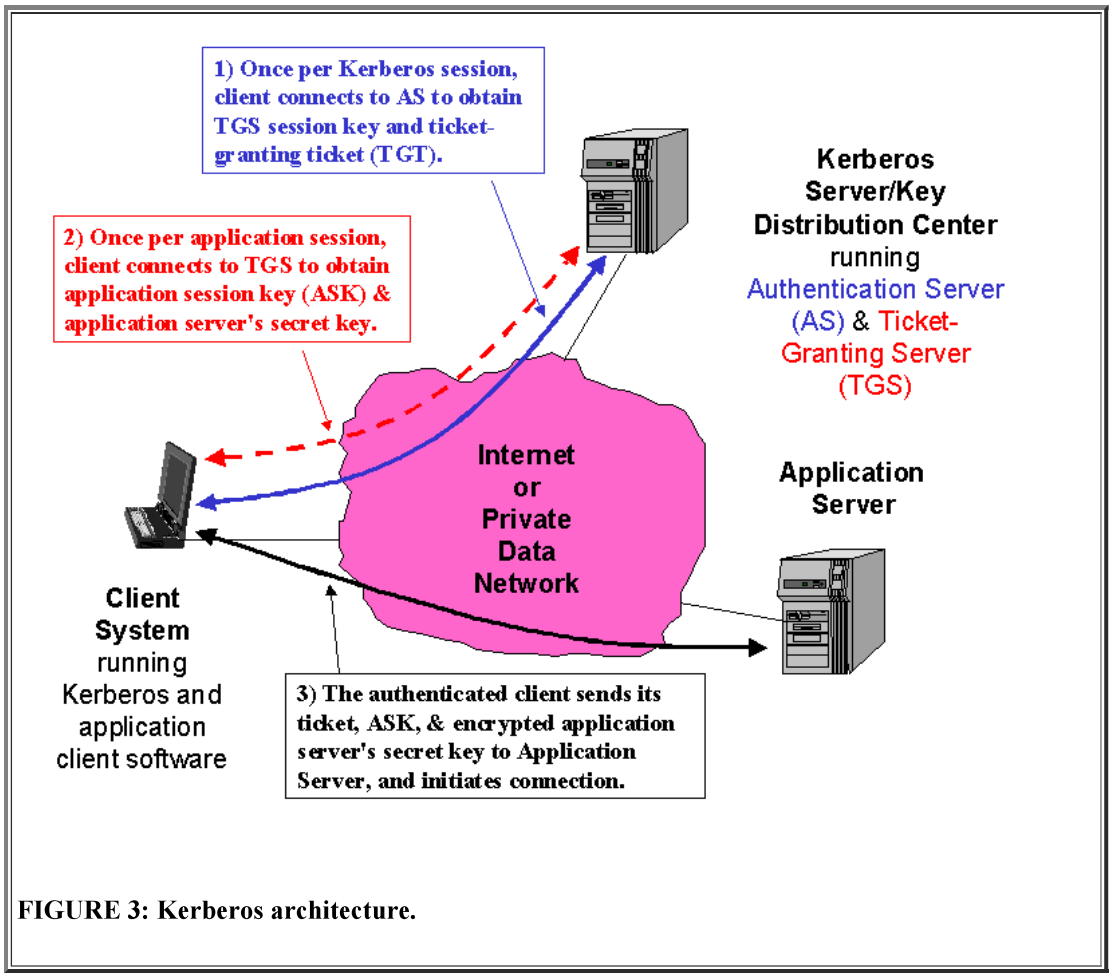


FIGURE 3: Kerberos architecture.

The Kerberos Server/KDC has two main functions (Figure 3), known as the Authentication Server (AS) and Ticket-Granting Server (TGS). The steps in establishing an authenticated session between an application client and the application server are:

1. The Kerberos client software establishes a connection with the Kerberos server's AS function. The AS first authenticates that the client is who it purports to be. The AS then provides the client with a secret key for this login session (the *TGS session key*) and a ticket-granting ticket (TGT), which gives the client permission to talk to the TGS. The ticket has a finite lifetime so that the authentication process is repeated periodically.
2. The client now communicates with the TGS to obtain the Application Server's key so that it (the client) can establish a connection to the service it wants. The client supplies the TGS with the TGS session key and TGT; the TGS responds with an application session key (ASK) and an encrypted form of the Application Server's secret key; this secret key is *never* sent on the network in any other form.
3. The client has now authenticated itself *and* can prove its identity to the Application Server by supplying the Kerberos ticket, application session key, and encrypted Application Server secret key. The Application Server responds with similarly encrypted information to authenticate itself to the client. At this point, the client can initiate the intended service requests (e.g., Telnet, FTP, HTTP, or e-commerce transaction session establishment).

The current shipping version of this protocol is Kerberos V5 (described in RFC 1510), although Kerberos V4 still exists and is seeing some use. While the details of their operation, functional capabilities, and message formats are different, the conceptual overview above pretty much holds for both. One primary difference is that Kerberos V4 uses only DES to generate keys and encrypt messages, while V5 allows other schemes to be employed (although DES is still the most widely algorithm used).

4.3. Public Key Certificates and Certificate Authorities

Certificates and *Certificate Authorities (CA)* are necessary for widespread use of cryptography for e-commerce applications. While a combination of secret and public key cryptography can solve the business issues discussed above, crypto cannot alone address the trust issues that must exist between a customer and vendor in the very fluid, very dynamic e-commerce relationship. How, for example, does one site obtain another party's public key? How does a recipient determine if a public key really belongs to the sender? How does the recipient know that the sender is using their public key for a legitimate purpose for which they are authorized? When does a public key expire? How can a key be revoked in case of compromise or loss?

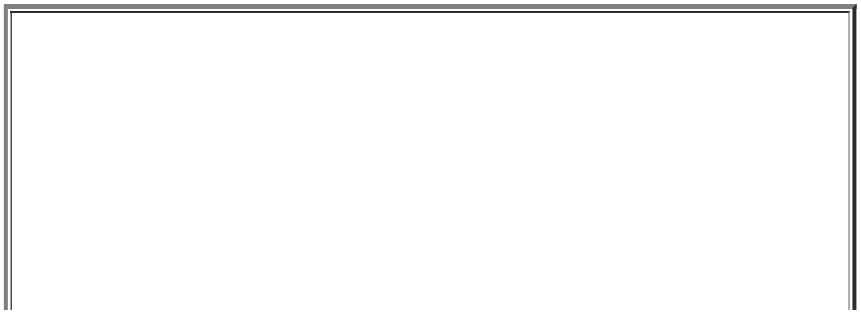
The basic concept of a certificate is one that is familiar to all of us. A driver's license, credit card, or SCUBA certification, for example, identify us to others, indicate something that we are authorized to do, have an expiration date, and identify the authority that granted the certificate.

As complicated as this may sound, it really isn't! Consider driver's licenses. I have one issued by the State of Vermont. The license establishes my identity, indicates the type of vehicles that I can operate and the fact that I must wear corrective lenses while doing so, identifies the issuing authority, and notes that I am an organ donor. When I drive outside of Vermont, the other jurisdictions throughout the U.S. recognize the authority of Vermont to issue this "certificate" and they trust the information it contains. Now, when I leave the U.S., everything changes. When I am in Canada and many other countries, they will accept not the Vermont license, per se, but *any* license issued in the U.S.; some other countries may not recognize the Vermont driver's license as sufficient bona fides that I can drive. This analogy represents the certificate chain, where even certificates carry certificates.

For purposes of electronic transactions, certificates are digital documents. The specific functions of the certificate include:

- *Establish identity:* Associate, or *bind*, a public key to an individual, organization, corporate position, or other entity.
- *Assign authority:* Establish what actions the holder may or may not take based upon this certificate.
- *Secure confidential information* (e.g., encrypting the session's symmetric key for data confidentiality).

Typically, a certificate contains a public key, a name, an expiration date, the name of the authority that issued the certificate (and, therefore, is vouching for the identity of the user), a serial number, any pertinent policies describing how the certificate was issued and/or how the certificate may be used, the digital signature of the certificate issuer, and perhaps other information.



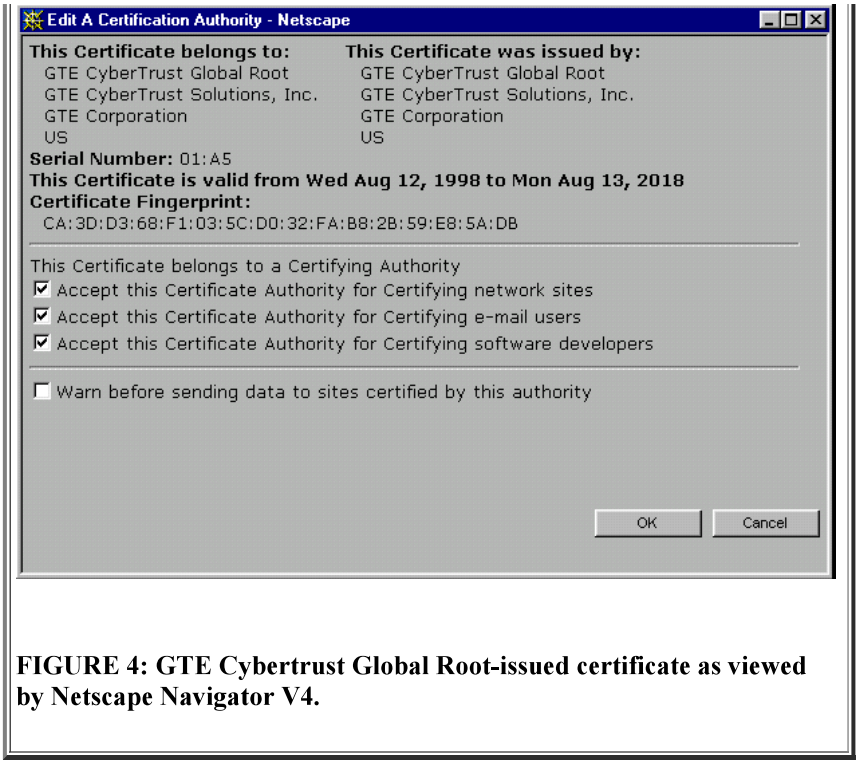


FIGURE 4: GTE Cybertrust Global Root-issued certificate as viewed by Netscape Navigator V4.

A sample abbreviated certificate is shown in Figure 4. This is a typical certificate found in a browser; while this one is issued by GTE Cybertrust, many so-called root-level certificates can be found shipped with browsers. When the browser makes a connection to a secure Web site, the Web server sends its public key certificate to the browser. The browser then checks the certificate's signature against the public key that it has stored; if there is a match, the certificate is taken as valid and the Web site verified by this certificate is considered to be "trusted."

TABLE 2. Contents of an X.509 V3 Certificate.

version number
certificate serial number
signature algorithm identifier
issuer's name and unique identifier
validity (or operational) period
subject's name and unique identifier
subject public key information
standard extensions
certificate appropriate use definition
key usage limitation definition
certificate policy information
other extensions
Application-specific
CA-specific

The most widely accepted certificate format is the one defined in International Telecommunication Union Telecommunication Standardization Sector (ITU-T) Recommendation X.509. Rec. X.509 is a specification used around the world and any applications complying with X.509 can share certificates. Most certificates today comply with X.509 Version 3 and contain the information listed in Table 2.

Certificate authorities are the repositories for public-keys and can be any agency that issues certificates. A company, for example, may issue certificates to its employees, a college/university to its students, a store to its customers, an Internet service provider to its users, or a government to its constituents.

When a sender needs an intended receiver's public key, the sender must get that key from the receiver's CA. That scheme is straight-forward if the sender and receiver have certificates issued by the same CA. If not, how does the sender know to *trust* the foreign CA? One industry wag has noted, about trust: "You are either born with it or have it granted upon you." Thus, some CAs will be trusted because they are known to be reputable, such as the CAs operated by AT&T, BBN, Canada Post Corp., CommerceNet, GTE Cybertrust, MCI, Nortel EnTrust, Thawte, the U.S. Postal Service, and VeriSign. CAs, in turn, form trust relationships with other CAs. Thus, if a user queries a foreign CA for information, the user may

ask to see a list of CAs that establish a "chain of trust" back to the user.

One major feature to look for in a CA is their identification policies and procedures. When a user generates a key pair and forwards the public key to a CA, the CA has to check the sender's identification and takes any steps necessary to assure itself that the request is really coming from the advertised sender. Different CAs have different identification policies and will, therefore, be trusted differently by other CAs. Verification of identity is just one of many issues that are part of a CA's Certification Practice Statement (CPS) and policies; other issues include how the CA protects the public keys in its care, how lost or compromised keys are revoked, and how the CA protects its own private keys.

4.4. Summary

The paragraphs above describe three very different trust models. It is hard to say that any one is better than the others; it depend upon your application. One of the biggest and fastest growing applications of cryptography today, though, is electronic commerce (e-commerce), a term that itself begs for a formal definition.

PGP's web of trust is easy to maintain and very much based on the reality of users as people. The model, however, is limited; just how many public keys can a single user reliably store and maintain? And what if you are using the "wrong" computer when you want to send a message and can't access your keyring? How easy it is to revoke a key if it is compromised? PGP may also not scale well to an e-commerce scenario of secure communication between total strangers on short-notice.

Kerberos overcomes many of the problems of PGP's web of trust, in that it is scalable and its scope can be very large. However, it also requires that the Kerberos server have *a priori* knowledge of all client systems prior to any transactions, which makes it unfeasible for "hit-and-run" client/server relationships as seen in e-commerce.

Certificates and the collection of CAs will form a Public Key Infrastructure (PKI). In the early days of the Internet, every host had to maintain a list of every other host; the Domain Name System (DNS) introduced the idea of a distributed database for this purpose and the DNS is one of the key reasons that the Internet has grown as it has. A PKI will fill a similar void in the e-commerce and PKC realm.

While certificates and the benefits of a PKI are most often associated with electronic commerce, the applications for PKI are much broader and include secure electronic mail, payments and electronic checks, Electronic Data Interchange (EDI), secure transfer of Domain Name System (DNS) and routing information, electronic forms, and digitally signed documents. A single "global PKI" is still many years away, that is the ultimate goal of today's work as international electronic commerce changes the way in which we do business in a similar way in which the Internet has changed the way in which we communicate.

5. CRYPTOGRAPHIC ALGORITHMS IN ACTION

The paragraphs above have provided an overview of the different types of cryptographic algorithms, as well as some examples of some available protocols and schemes. Table 3 provides a list of some other noteworthy schemes employed — or proposed — for a variety of functions, most notably electronic commerce. The paragraphs below will show several real cryptographic applications that many of us employ (knowingly or not) everyday for password protection and private communication.

TABLE 3. Other Crypto Algorithms and Systems of Note.

Capstone	A now-defunct U.S. National Institute of Standards and Technology (NIST) and National Security Agency (NSA) project under the Bush Sr. and Clinton administrations for publicly available strong cryptography with keys escrowed by the government (NIST and the Treasury Dept.). Capstone included in one or more tamper-proof computer chips for implementation (Clipper), a secret key encryption algorithm (Skipjack), digital signature algorithm (DSA), key exchange algorithm (KEA), and hash algorithm (SHA).
Clipper	The computer chip that would implement the Skipjack encryption scheme. See also EPIC's The Clipper Chip Web page.
Derived Unique Key Per Transaction (DUKPT)	A key management scheme used for debit and credit card verification with point-of-sale (POS) transaction systems, automated teller machines (ATMs), and other financial applications. In DUKPT, a unique key is derived for each transaction based upon a fixed, shared key in such a way that knowledge of one derived key does not easily yield knowledge of

	other keys (including the fixed key). Therefore, if one of the derived keys is compromised, neither past nor subsequent transactions are endangered. DUKPT is specified in American National Standard (ANS) ANSI X9.24-1:2009 *Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques*) and can be purchased at the ANSI Web page.
Escrowed Encryption Standard (EES)	Largely unused, a controversial crypto scheme employing the SKIPJACK secret key crypto algorithm and a Law Enforcement Access Field (LEAF) creation method. LEAF was one part of the key escrow system and allowed for decryption of ciphertext messages that had been legally intercepted by law enforcement agencies. Described more in FIPS 185.
Federal Information Processing Standards (FIPS)	These computer security- and crypto-related FIPS are produced by the U.S. National Institute of Standards and Technology (NIST) as standards for the U.S. Government.
Fortezza (formerly called Tessera)	A PCMCIA card developed by NSA that implements the Capstone algorithms, intended for use with the Defense Messaging Service (DMS).
GOST	GOST is a family of algorithms that is defined in the Russian cryptographic standards. Although most of the specifications are written in Russian, a series of RFCs describe some of the aspects so that the algorithms can be used effectively in Internet applications: • RFC 4357: Additional Cryptographic Algorithms for Use with GOST 28147-89, GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms • RFC 5830: GOST 28147-89: Encryption, Decryption, and Message Authentication Code (MAC) Algorithms • RFC 6986: GOST R 34.11-2012: Hash Function Algorithm • RFC 7091: GOST R 34.10-2012: Digital Signature Algorithm (Updates RFC 5832: GOST R 34.10-2001)
Identity-Based Encryption (IBE)	Identity-Based Encryption was first proposed by Adi Shamir in 1984, and is a key authentication system where the public key can be derived from some unique information based upon the user's identity. In 2001, Dan Boneh (Stanford) and Matt Franklin (U.C., Davis) developed a practical implementation of IBE based on elliptic curves and a mathematical construct called the Weil Pairing. In that year, Clifford Cocks (GCHQ) also described another IBE solution based on quadratic residues in composite groups. • RFC 5091: Identity-Based Cryptography Standard (IBCS) #1: Describes an implementation of IBE using Boneh-Franklin (BF) and Boneh-Boyen (BB1) Identity-based Encryption. This document is in part based on Voltage Security's Identity-based Cryptography Standards (IBCS) documents.
IP Security Protocol (IPsec)	The IPsec protocol suite is used to provide privacy and authentication services at the IP layer. An overview of the protocol suite and of the documents comprising IPsec can be found in RFC 2411. Other documents include: • RFC 4301: IP security architecture. • RFC 4302: IP Authentication Header (AH), one of the two primary IPsec functions; AH provides connectionless integrity and data origin authentication for IP datagrams and protects against replay attacks. • RFC 4303: IP Encapsulating Security Payload (ESP), the other primary IPsec function; ESP provides a variety of security services within IPsec. • RFC 4304: Extended Sequence Number (ESN) Addendum, allows for negotiation of a 32- or 64- bit sequence number, used to detect replay attacks. • RFC 4305: Cryptographic algorithm implementation

	requirements for ESP and AH. • RFC 5996: The Internet Key Exchange (IKE) protocol, version 2, providing for mutual authentication and establishing and maintaining security associations. ◦ IKE v1 was described in three separate documents, RFC 2407 (application of ISAKMP to IPsec), RFC 2408 (ISAKMP, a framework for key management and security associations), and RFC 2409 (IKE, using part of Oakley and part of SKEME in conjunction with ISAKMP to obtain authenticated keying material for use with ISAKMP, and for other security associations such as AH and ESP). IKE v1 is obsoleted with the introdcution of IKEv2. • RFC 4307: Cryptographic algoritms used with IKEv2. • RFC 4308: Crypto suites for IPsec, IKE, and IKEv2. • RFC 4309: The use of AES in CBC-MAC mode with IPsec ESP. • RFC 4312: The use of the Camellia cipher algorithm in IPsec. • RFC 4359: The Use of RSA/SHA-1 Signatures within Encapsulating Security Payload (ESP) and Authentication Header (AH). • RFC 4434: Describes AES-XCBC-PRF-128, a pseudo-random function derived from the AES for use with IKE. • RFC 2403: Describes use of the HMAC with MD5 algorithm for data origin authentication and integrity protection in both AH and ESP. • RFC 2405: Describes use of DES-CBC (DES in Cipher Block Chaining Mode) for confidentiality in ESP. • RFC 2410: Defines use of the NULL encryption algorithm (i.e., provides authentication and integrity without confidentiality) in ESP. • RFC 2412: Describes OAKLEY, a key determination and distribution protocol. • RFC 2451: Describes use of Cipher Block Chaining (CBC) mode cipher algorithms with ESP. • RFCs 2522 and 2523: Description of Photuris, a session-key management protocol for IPsec. In addition, RFC 6379 describes Suite B Cryptographic Suites for IPsec and RFC 6380 describes the Suite B profile for IPsec. IPsec was first proposed for use with IP version 6 (IPv6), but can also be employed with the current IP version, IPv4. (See more detail about IPsec below in Section 5.6.)
Internet Security Association and Key Management Protocol (ISAKMP/OAKLEY)	ISAKMP/OAKLEY provide an infrastructure for Internet secure communications. ISAKMP, designed by the National Security Agency (NSA) and described in RFC 2408, is a framework for key management and security associations, independent of the key generation and cryptographic algorithms actually employed. The OAKLEY Key Determination Protocol, described in RFC 2412, is a key determination and distribution protocol using a variation of Diffie-Hellman.
Kerberos	A secret-key encryption and authentication system, designed to authenticate requests for network resources within a user domain rather than to authenticate messages. Kerberos also uses a trusted third-party approach; a client communications with the Kerberos server to obtain "credentials" so that it may access services at the application server. Kerberos V4 uses DES to generate keys and encrypt messages; DES is also commonly used in Kerberos V5, although other schemes could be employed. Microsoft added support for Kerberos V5 — with some proprietary extensions — in Windows 2000. There are many Kerberos articles posted at Microsoft's Knowledge Base, notably "Basic Overview of Kerberos User Authentication Protocol in Windows 2000," "Windows 2000 Kerberos 5 Ticket Flags and KDC Options for

	AS_REQ and TGS_REQ Messages," and "Kerberos Administration in Windows 2000."
Keyed-Hash Message Authentication Code (HMAC)	A message authentication scheme based upon secret key cryptography and the secret key shared between two parties rather than public key methods. Described in FIPS 198 and RFC 2104.
Message Digest Cipher (MDC)	Invented by Peter Gutman, MDC turns a one-way hash function into a block cipher.
MIME Object Security Standard (MOSS)	Designed as a successor to PEM to provide PEM-based security services to MIME messages.
NSA Suite B Cryptography	An NSA standard for securing information at the SECRET level. Defines use of: • Advanced Encryption Standard (AES) with key sizes of 128 and 256 bits, per FIPS PUB 197 for encryption • The Ephemeral Unified Model and the One-Pass Diffie Hellman (referred to as ECDH) using the curves with 256 and 384-bit prime moduli, per NIST Special Publication 800-56A for key exchange • Elliptic Curve Digital Signature Algorithm (ECDSA) using the curves with 256 and 384-bit prime moduli, per FIPS PUB 186-3 for digital signatures • Secure Hash Algorithm (SHA) using 256 and 384 bits, per FIPS PUB 180-3 for hashing RFC 6239 describes Suite B Cryptographic Suites for Secure Shell (SSH) and RFC 6379 describes Suite B Cryptographic Suites for Secure IP (IPsec).
Pretty Good Privacy (PGP)	A family of cryptographic routines for e-mail and file storage applications developed by Philip Zimmermann. PGP 2.6.x uses RSA for key management and digital signatures, IDEA for message encryption, and MD5 for computing the message's hash value; more information can also be found in RFC 1991. PGP 5.x (formerly known as "PGP 3") uses Diffie-Hellman/DSS for key management and digital signatures; IDEA, CAST, or 3DES for message encryption; and MD5 or SHA for computing the message's hash value. OpenPGP, described in RFC 2440, is an open definition of security software based on PGP 5.x. (See more detail about PGP below in Section 5.5.)
Privacy Enhanced Mail (PEM)	Provides secure electronic mail over the Internet and includes provisions for encryption (DES), authentication, and key management (DES, RSA). May be superseded by S/MIME and PEM-MIME. Developed by IETF PEM Working Group and defined in four RFCs: • RFC 1421: Part I, Message Encryption and Authentication Procedures • RFC 1422: Part II, Certificate-Based Key Management • RFC 1423: Part III, Algorithms, Modes, and Identifiers • RFC 1424: Part IV, Key Certification and Related Services
Private Communication Technology (PCT)	Developed by Microsoft and Visa for secure communication on the Internet. Similar to SSL, PCT supports Diffie-Hellman, Fortezza, and RSA for key establishment; DES, RC2, RC4, and triple-DES for encryption; and DSA and RSA message signatures. A companion to SET.
Secure Electronic Transactions (SET)	A merging of two other protocols: SEPP (Secure Electronic Payment Protocol), an open specification for secure bank card transactions over the Internet, developed by CyberCash, GTE, IBM, MasterCard, and Netscape; and STT (Secure Transaction Technology), a secure payment protocol developed by Microsoft and Visa International. Supports DES and RC4 for encryption, and RSA for signatures, key exchange, and public-key encryption of

	bank card numbers. SET is a companion to the PCT protocol.
Secure Hypertext Transfer Protocol (S-HTTP)	An extension to HTTP to provide secure exchange of documents over the World Wide Web. Supported algorithms include RSA and Kerberos for key exchange, DES, IDEA, RC2, and Triple-DES for encryption.
Secure Multipurpose Internet Mail Extensions (S/MIME)	An IETF secure e-mail scheme intended to supercede PEM. S/MIME, described in RFCs 2311 and 2312, adds digital signature and encryption capability to Internet MIME messages.
Secure Sockets Layer (SSL)	Developed by Netscape Communications to provide application-independent security and privacy over the Internet. SSL is designed so that protocols such as HTTP, FTP (File Transfer Protocol), and Telnet can operate over it transparently. SSL allows both server authentication (mandatory) and client authentication (optional). RSA is used during negotiation to exchange keys and identify the actual cryptographic algorithm (DES, IDEA, RC2, RC4, or 3DES) to use for the session. SSL also uses MD5 for message digests and X.509 public-key certificates. SSL was found to be breakable soon after the IETF announced formation of group to work on TLS and RFC 6176 specifically prohibits the use of SSL v2.0 by TLS clients. SSL version 3.0 is described in RFC 6101. (See more detail about SSL below in Section 5.7.)
Server Gated Cryptography (SGC)	Microsoft extension to SSL that provides strong encryption for online banking and other financial applications using RC2 (128-bit key), RC4 (128-bit key), DES (56-bit key), or 3DES (equivalent of 168-bit key). Use of SGC requires a Windows NT Server running Internet Information Server (IIS) 4.0 with a valid SGC certificate. SGC is available in 32-bit Windows versions of Internet Explorer (IE) 4.0, and support for Mac, Unix, and 16-bit Windows versions of IE is expected in the future.
Simple Authentication and Security Layer (SASL)	(SASL) is a framework for providing authentication and data security services in connection-oriented protocols (a la TCP). It provides a structured interface and allows new protocols to reuse existing authentication mechanisms and allows old protocols to make use of new mechanisms. It has been common practice on the Internet to permit anonymous access to various services, employing a plain-text password using a user name of "anonymous" and a password of an email address or some other identifying information. New IETF protocols disallow plain-text logins. The Anonymous SASL Mechanism (RFC 4505) provides a method for anonymous logins within the SASL framework.
Simple Key-Management for Internet Protocol (SKIP)	Key management scheme for secure IP communication, specifically for IPsec, and designed by Aziz and Diffie. SKIP essentially defines a public key infrastructure for the Internet and even uses X.509 certificates. Most public key cryptosystems assign keys on a per-session basis, which is inconvenient for the Internet since IP is connectionless. Instead, SKIP provides a basis for secure communication between any pair of Internet hosts. SKIP can employ DES, 3DES, IDEA, RC2, RC5, MD5, and SHA-1.
Transport Layer Security (TLS)	TLS v1.0 is an IETF specification (RFC 2246) intended to replace SSL. TLS v1.0 employs Triple-DES (secret key cryptography), SHA (hash), Diffie-Hellman (key exchange), and DSS (digital signatures). TLS v1.0 was vulnerable to attack and updated by v1.1 (RFC 4346) and v1.2 (RFC 5246); v1.3 is currently a draft specification. TLS is designed to operate over TCP. The IETF developed the Datagram Transport Layer Security (DTLS) protocol to operate over UDP. DTLS v1.2 is described in RFC 6347. (See more detail about TLS below in Section 5.7.)

TrueCrypt	Open source, multi-platform cryptography software that can be used to encrypt a file, partition, or entire disk. One of TrueCrypt's more interesting features is that of *plausible deniability* with hidden volumes or hidden operating systems. To be replaced by CipherShed. (See more detail about TrueCrypt below in Section 5.11.)
X.509	ITU-T recommendation for the format of certificates for the public key infrastructure. Certificates map (bind) a user identity to a public key. The IETF application of X.509 certificates is documented in RFC 2459. An Internet X.509 Public Key Infrastructure is further defined in RFC 2510 (Certificate Management Protocols) and RFC 2527 (Certificate Policy and Certification Practices Framework).

5.1. Password Protection

Nearly all modern multiuser computer and network operating systems employ passwords at the very least to protect and authenticate users accessing computer and/or network resources. But passwords are *not* typically kept on a host or server in plaintext, but are generally encrypted using some sort of hash scheme.

```
A) /etc/passwd file

 root:Jbw6BwE4XoUHo:0:0:root:/root:/bin/bash
 carol:FM5ikbQt1K052:502:100:Carol Monaghan:/home/carol:/bin/bash
 alex:LqAi7Mdyg/HcQ:503:100:Alex Insley:/home/alex:/bin/bash
 gary:FkJXupRyFqY4s:501:100:Gary Kessler:/home/gary:/bin/bash
 todd:edGqQUAaGv7g6:506:101:Todd Pritsky:/home/todd:/bin/bash
 josh:FiH0ONcjPut1g:505:101:Joshua Kessler:/home/webroot:/bin/bash

B.1) /etc/passwd file (with shadow passwords)

 root:x:0:0:root:/root:/bin/bash
 carol:x:502:100:Carol Monaghan:/home/carol:/bin/bash
 alex:x:503:100:Alex Insley:/home/alex:/bin/bash
 gary:x:501:100:Gary Kessler:/home/gary:/bin/bash
 todd:x:506:101:Todd Pritsky:/home/todd:/bin/bash
 josh:x:505:101:Joshua Kessler:/home/webroot:/bin/bash

B.2) /etc/shadow file

 root:AGFw$1$P4u/uhLK$l2.HP35rlu65WlfCzq:11449:0:99999:7:::
 carol:kjHaN%35a8xMM8a/0kMl1?fwtLAM.K&kw.:11449:0:99999:7:::
 alex:1$1KKmfTy0a7#3.LL9a8H71lkwn/.hH22a:11449:0:99999:7:::
 gary:9ajlknknKJHjhnu7298ypnAIJKL$Jh.hnk:11449:0:99999:7:::
 todd:798POJ90uab6.k$klPqMt%alMlprWqu6$.:11492:0:99999:7:::
 josh:Awmqpsui*787pjnsnJJK%aappaMpQo07.8:11492:0:99999:7:::
```

FIGURE 5: Sample entries in Unix/Linux password files.

Unix/Linux, for example, uses a well-known hash via its *crypt()* function. Passwords are stored in the */etc/passwd* file (Figure 5A); each record in the file contains the username, hashed password, user's individual and group numbers, user's name, home directory, and shell program; these fields are separated by colons (:). Note that each password is stored as a 13-byte string. The first two characters are actually a *salt*, randomness added to each password so that if two users have the same password, they will still be encrypted differently; the salt, in fact, provides a means so that a single password might have 4096 different encryptions. The remaining 11 bytes are the password hash, calculated using DES.

As it happens, the */etc/passwd* file is world-readable on Unix systems. This fact, coupled with the weak encryption of the passwords, resulted in the development of the *shadow password* system where passwords are kept in a separate, non-world-readable file used in conjunction with the normal password file. When shadow passwords are used, the password entry in */etc/passwd* is replaced with a "*" or "x" (Figure 5B.1) and the MD5 hash of the passwords are stored in */etc/shadow* along with some other account information (Figure 5B.2).

Windows NT uses a similar scheme to store passwords in the Security Access Manager (SAM) file. In the NT case, all passwords are hashed using the MD4 algorithm, resulting in a 128-bit (16-byte) hash value (they are then *obscured* using an undocumented mathematical transformation that was a secret until distributed on the Internet). The password

`password`, for example, might be stored as the hash value (in hexadecimal) `60771b22d73c34bd4a290a79c8b09f18`.

Passwords are not saved in plaintext on computer systems precisely so they cannot be easily compromised. For similar reasons, we don't want passwords sent in plaintext across a network. But for remote logon applications, how does a client system identify itself or a user to the server? One mechanism, of course, is to send the password as a hash value and that, indeed, may be done. A weakness of that approach, however, is that an intruder can grab the password off of the network and use an off-line attack (such as a *dictionary attack* where an attacker takes every known word and encrypts it with the network's encryption algorithm, hoping eventually to find a match with a purloined password hash). In some situations, an attacker only has to copy the hashed password value and use it later on to gain unauthorized entry without ever learning the actual password.

An even stronger authentication method uses the password to modify a shared secret between the client and server, but never allows the password in any form to go across the network. This is the basis for the Challenge Handshake Authentication Protocol (CHAP), the remote logon process used by Windows NT.

As suggested above, Windows NT passwords are stored in a security file on a server as a 16-byte hash value. In truth, Windows NT stores *two* hashes; a weak hash based upon the old LAN Manager (LanMan) scheme and the newer NT hash. When a user logs on to a server from a remote workstation, the user is identified by the username, sent across the network in plaintext (no worries here; it's not a secret anyway!). The server then generates a 64-bit random number and sends it to the client (also in plaintext). This number is the *challenge*.

Using the LanMan scheme, the client system then encrypts the challenge using DES. Recall that DES employs a 56-bit key, acts on a 64-bit block of data, and produces a 64-bit output. In this case, the 64-bit data block is the random number. The client actually uses three different DES keys to encrypt the random number, producing three different 64-bit outputs. The first key is the first seven bytes (56 bits) of the password's hash value, the second key is the next seven bytes in the password's hash, and the third key is the remaining two bytes of the password's hash concatenated with five zero-filled bytes. (So, for the example above, the three DES keys would be `60771b22d73c34`, `bd4a290a79c8b0`, and `9f180000000000`.) Each key is applied to the random number resulting in three 64-bit outputs, which comprise the *response*. Thus, the server's 8-byte challenge yields a 24-byte response from the client and this is all that would be seen on the network. The server, for its part, does the same calculation to ensure that the values match.

There is, however, a significant weakness to this system. Specifically, the response is generated in such a way as to effectively reduce 16-byte hash to three smaller hashes, of length seven, seven, and two. Thus, a password cracker has to break at most a 7-byte hash. One Windows NT vulnerability test program that I have used in the past will report passwords that are "too short," defined as "less than 8 characters." When I asked how the program knew that passwords were too short, the software's salespeople suggested to me that the program broke the passwords to determine their length. This is undoubtedly not true; all the software really has to do is look at the second 7-byte block and some known value indicates that it is empty, which would indicate a password of seven or less characters.

Consider the following example, showing the LanMan hash of two different short passwords (take a close look at the last 8 bytes):

AA: `89D42A44E77140AAAAD3B435B51404EE`
AAA: `1C3A2B6D939A1021AAD3B435B51404EE`

Note that the NT hash provides no such clue:

AA: `C5663434F963BE79C8FD99F535E7AAD8`
AAA: `6B6E0FB2ED246885B98586C73B5BFB77`

It is worth noting that the discussion above describes the Microsoft version of CHAP, or MS-CHAP (MS-CHAPv2 is described in RFC 2759). MS-CHAP assumes that it is working with hashed values of the password as the key to encrypting the challenge. More traditional CHAP (RFC 1994) assumes that it is starting with passwords in plaintext. The relevance of this observation is that a CHAP client, for example, cannot be authenticated by an MS-CHAP server; both client and server must use the same CHAP version.

5.2. Some of the Finer Details of Diffie-Hellman

Diffie and Hellman introduced the concept of public-key cryptography. The mathematical "trick" of Diffie-Hellman key exchange is that it is relatively easy to compute exponents compared to computing discrete logarithms. Diffie-Hellman allows two parties — the ubiquitous Alice and Bob — to generate a secret key; they need to exchange some information over an unsecure communications channel to perform the calculation but an eavesdropper cannot determine the shared secret key based upon this information.

Diffie-Hellman works like this. Alice and Bob start by agreeing on a large prime number, N. They also have to choose some number G so that G<N.

There is actually another constraint on G, namely that it must be primitive with respect to N. *Primitive* is a definition that is a little beyond the scope of our discussion but basically G is primitive to N if we can find integers *i* so that G^i mod N = j for all values of j from 1 to N-1. As an example, 2 is not primitive to 7 because the set of powers of 2 from 1 to 6, mod 7 (i.e., 2^1 mod 7, 2^2 mod 7 ... 2^6 mod 7) = {2,4,1,2,4,1}. On the other hand, 3 is primitive to 7 because the set of powers of 3 from 1 to 6, mod 7 = {3,2,6,4,5,1}.

(The definition of primitive introduced a new term to some readers, namely *mod*. The phrase *x mod y* (and read as written!) means "take the remainder after dividing x by y." Thus, 1 mod 7 = 1, 9 mod 6 = 3, and 8 mod 8 = 0. Read more about the *modulo* function in the appendix.)

Anyway, either Alice or Bob selects N and G; they then tell the other party what the values are. Alice and Bob then work independently:

Alice...

1. Choose a large random number, $X_A < N$. This is Alice's private key.
2. Compute $Y_A = G^{X_A}$ mod N. This is Alice's public key.
3. Exchange public key's with Bob.
4. Compute $K_A = Y_B^{X_A}$ mod N

Bob...

1. Choose a large random number, $X_B < N$. This is Bob's private key.
2. Compute $Y_B = G^{X_B}$ mod N. This is Bob's public key.
3. Exchange public key's with Alice.
4. Compute $K_B = Y_A^{X_B}$ mod N

Note that X_A and X_B are kept secret while Y_A and Y_B are openly shared; these are the private and public keys, respectively. Based on their own private key and the public key learned from the other party, Alice and Bob have computed their secret keys, K_A and K_B, respectively, which are equal to $G^{X_A X_B}$ mod N.

Perhaps a small example will help here. Although Alice and Bob will really choose large values for N and G, I will use small values for example only; let's use N=7 and G=3.

Alice...

1. Choose $X_A = 2$
2. Calculate $Y_A = 3^2$ mod 7 = 2
3. Exchange public keys with Bob
4. $K_A = 6^2$ mod 7 = 1

Bob...

1. Choose $X_B = 3$
2. Calculate $Y_B = 3^3$ mod 7 = 6
3. Exchange public keys with Alice
4. $K_B = 2^3$ mod 7 = 1

In this example, then, Alice and Bob will both find the secret key 1 which is, indeed, 3^6 mod 7 (i.e., $G^{X_A X_B} = 3^{2*3}$). If an eavesdropper (Mallory) was listening in on the information exchange between Alice and Bob, he would learn G, N, Y_A, and Y_B which is a lot of information but insufficient to compromise the key; as long as X_A and X_B remain unknown, K is safe. As said above, calculating $Y = G^X$ is a lot easier than finding $X = \log_G Y$.

> **A short digression on modulo arithmetic.** In the paragraph above, we noted that 3^6 mod 7 = 1. This can be confirmed, of course, by noting that:
>
> $$3^6 = 729 = 104*7 + 1$$
>
> There is a nice property of modulo arithmetic, however, that makes this determination a little easier, namely: (a mod x)(b mod x) = (ab mod x). Therefore, one possible shortcut is to note that $3^6 = (3^3)(3^3)$. Therefore, 3^6 mod 7 = (3^3 mod 7)(3^3 mod 7) = (27 mod 7)(27 mod 7) = 6*6 mod 7 = 36 mod 7 = 1.

Diffie-Hellman can also be used to allow key sharing amongst multiple users. Note again that the Diffie-Hellman algorithm is used to generate secret keys, not to encrypt and decrypt messages.

5.3. Some of the Finer Details of RSA Public-Key Cryptography

Unlike Diffie-Hellman, RSA can be used for key exchange as well as digital signatures and the encryption of small blocks of data. Today, RSA is primarily used to encrypt the session key used for secret key encryption (message integrity) or the message's hash value (digital signature). RSA's mathematical hardness comes from the ease in calculating large numbers and the difficulty in finding the prime factors of those large numbers. Although employed with numbers using hundreds of digits, the math behind RSA is relatively straight-forward.

To create an RSA public/private key pair, here are the basic steps:

1. Choose two prime numbers, p and q. From these numbers you can calculate the modulus, n = pq.
2. Select a third number, e, that is relatively prime to (i.e., it does not divide evenly into) the product (p-1)(q-1). The number e is the public exponent.
3. Calculate an integer d from the quotient (ed-1)/[(p-1)(q-1)]. The number d is the private exponent.

The public key is the number pair (n,e). Although these values are publicly known, it is computationally infeasible to determine d from n and e if p and q are large enough.

To encrypt a message, M, with the public key, create the ciphertext, C, using the equation:

$$C = M^e \bmod n$$

The receiver then decrypts the ciphertext with the private key using the equation:

$$M = C^d \bmod n$$

Now, this might look a bit complex and, indeed, the mathematics does take a lot of computer power given the large size of the numbers; since p and q may be 100 digits (decimal) or more, d and e will be about the same size and n may be over 200 digits. Nevertheless, a simple example may help. In this example, the values for p, q, e, and d are purposely chosen to be very small and the reader will see exactly how badly these values perform, but hopefully the algorithm will be adequately demonstrated:

1. Select p=3 and q=5.
2. The modulus n = pq = 15.
3. The value e must be relatively prime to (p-1)(q-1) = (2)(4) = 8. Select e=11
4. The value d must be chosen so that (ed-1)/[(p-1)(q-1)] is an integer. Thus, the value (11d-1)/[(2)(4)] = (11d-1)/8 must be an integer. Calculate one possible value, d=3.
5. Let's say we wish to send the string **SECRET**. For this example, we will convert the string to the decimal representation of the ASCII values of the characters, which would be **83 69 67 82 69 84**.
6. The sender encrypts each digit one at a time (we have to because the modulus is so small) using the public key value (e,n)=(11,15). Thus, each ciphertext character $C_i = M_i^{11} \bmod 15$. The input digit string **0x836967826984** will be transmitted as **0x2c696d286924**.
7. The receiver decrypts each digit using the private key value (d,n)=(3,15). Thus, each plaintext character $M_i = C_i^3 \bmod 15$. The input digit string **0x2c696d286924** will be converted to **0x836967826984** and, presumably, reassembled as the plaintext string **SECRET**.

Again, the example above uses small values for simplicity and, in fact, shows the weakness of small values; note that 4, 6, and 9 do not change when encrypted, and that the values 2 and 8 encrypt to 8 and 2, respectively. Nevertheless, this simple example demonstrates how RSA can be used to exchange information.

RSA keylengths of 512 and 768 bits are considered to be pretty weak. The minimum suggested RSA key is 1024 bits; 2048 and 3072 bits are even better.

As an aside, Adam Back (http://www.cypherspace.org/~adam/) wrote a two-line Perl script to implement RSA. It employs `dc`, an arbitrary precision arithmetic package that ships with most UNIX systems:

```
print pack"C*",split/\D+/,`echo "16iII*o\U@{$/=$z;[(pop,pop,unpack"H*",<>
)]}\EsMsKsN0[lN*1lK[d2%Sa2/d0<X+d*lMLa^*lN%0]dsXx++lMlN/dsM0<J]dsJxp"|dc`
```

5.4. Some of the Finer Details of DES, Breaking DES, and DES Variants

The Data Encryption Standard (DES) started life in the mid-1970s, adopted by the National Bureau of Standards (NBS) [now the National Institute for Standards and Technology (NIST)] as Federal Information Processing Standard 46 (FIPS 46-3) and by the American National Standards Institute (ANSI) as X3.92.

As mentioned earlier, DES uses the Data Encryption Algorithm (DEA), a secret key block-cipher employing a 56-bit key operating on 64-bit blocks. FIPS 81 describes four modes of DES operation: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB). Despite all of these options, ECB is the most commonly deployed mode of operation.

NIST finally declared DES obsolete in 2004, and withdrew FIPS 46-3, 74, and 81 (*Federal Register*, July 26, 2004, *69*(142), 44509-44510). Although other block ciphers have replaced DES, it is still interesting to see how DES encryption is performed; not only is it sort of neat, but DES was the first crypto scheme commonly seen in non-governmental applications and was the catalyst for modern "public" cryptography and the first public Feistel cipher. DES still remains in many products — and cryptography students and cryptographers will continue to study DES for years to come.

DES Operational Overview

DES uses a 56-bit key. In fact, the 56-bit key is divided into eight 7-bit blocks and an 8th odd parity bit is added to each block (i.e., a "0" or "1" is added to the block so that there are an odd number of 1 bits in each 8-bit block). By using the 8 parity bits for rudimentary error detection, a DES key is actually 64 bits in length for computational purposes although it only has 56 bits worth of randomness, or *entropy* (See Section A.3 for a brief discussion of entropy and information theory).

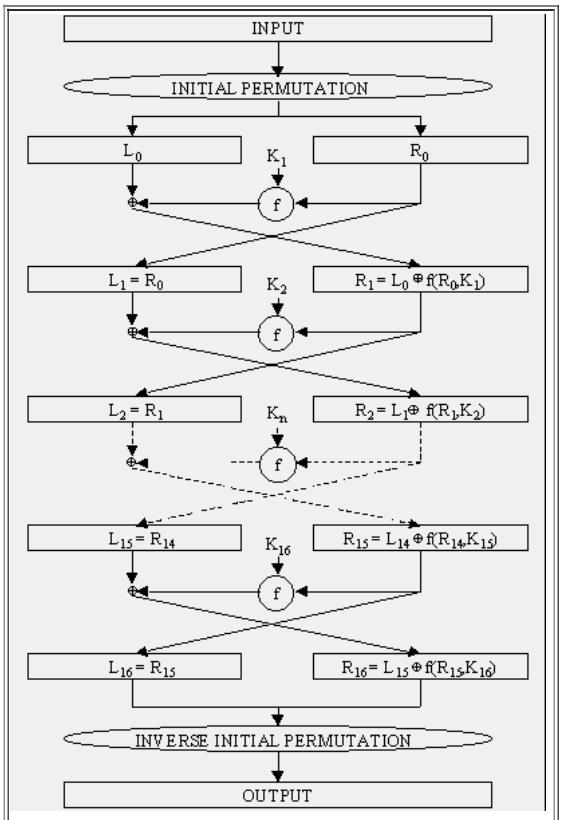


FIGURE 6: DES enciphering algorithm.

DES then acts on 64-bit blocks of the plaintext, invoking 16 rounds of permutations, swaps, and substitutes, as shown in Figure 6. The standard includes tables describing all of the selection, permutation, and expansion operations mentioned below; these aspects of the algorithm are not secrets. The basic DES steps are:

1. The 64-bit block to be encrypted undergoes an initial permutation (IP), where each bit is moved to a new bit position; e.g., the 1st, 2nd, and 3rd bits are moved to the 58th, 50th, and 42nd position, respectively.

2. The 64-bit permuted input is divided into two 32-bit blocks, called *left* and *right*, respectively. The initial values of the left and right blocks are denoted L_0 and R_0.

3. There are then 16 rounds of operation on the L and R blocks. During each iteration (where *n* ranges from 1 to 16), the following formulae apply:
   $L_n = R_{n-1}$
   $R_n = L_{n-1}$ XOR $f(R_{n-1},K_n)$
   At any given step in the process, then, the new L block value is merely taken from the prior R block value. The new

R block is calculated by taking the bit-by-bit exclusive-OR (XOR) of the prior L block with the results of applying the DES cipher function, *f*, to the prior R block and K_n. (K_n is a 48-bit value derived from the 64-bit DES key. Each round uses a different 48 bits according to the standard's Key Schedule algorithm.)

The cipher function, f, combines the 32-bit R block value and the 48-bit subkey in the following way. First, the 32 bits in the R block are expanded to 48 bits by an expansion function (E); the extra 16 bits are found by repeating the bits in 16 predefined positions. The 48-bit expanded R-block is then ORed with the 48-bit subkey. The result is a 48-bit value that is then divided into eight 6-bit blocks. These are fed as input into 8 selection (S) boxes, denoted $S_1,...,S_8$. Each 6-bit input yields a 4-bit output using a table lookup based on the 64 possible inputs; this results in a 32-bit output from the S-box. The 32 bits are then rearranged by a permutation function (P), producing the results from the cipher function.

4. The results from the final DES round — i.e., L_{16} and R_{16} — are recombined into a 64-bit value and fed into an inverse initial permutation (IP^{-1}). At this step, the bits are rearranged into their original positions, so that the 58th, 50th, and 42nd bits, for example, are moved back into the 1st, 2nd, and 3rd positions, respectively. The output from IP^{-1} is the 64-bit ciphertext block.

Consider this example with the given 56-bit key and input:

Key: **1100101 0100100 1001001 0011101 0110101 0101011 1101100 0011010**

Input character string: **GoAggies**
Input bit string: **11100010 11110110 10000010 11100110 11100110 10010110 10100110 11001110**

Output bit string: **10011111 11110010 10000000 10000001 01011011 00101001 00000011 00101111**
Output character string: **ùOÚ"Àô**

Breaking DES

The mainstream cryptographic community has long held that DES's 56-bit key was too short to withstand a brute-force attack from modern computers. Remember Moore's Law: computer power doubles every 18 months. Given that increase in power, a key that could withstand a brute-force guessing attack in 1975 could hardly be expected to withstand the same attack a quarter century later.

DES is even more vulnerable to a brute-force attack because it is often used to encrypt words, meaning that the entropy of the 64-bit block is, effectively, greatly reduced. That is, if we are encrypting random bit streams, then a given byte might contain any one of 2^8 (256) possible values and the entire 64-bit block has 2^{64}, or about 18.5 quintillion, possible values. If we are encrypting words, however, we are most likely to find a limited set of bit patterns; perhaps 70 or so if we account for upper and lower case letters, the numbers, space, and some punctuation. This means that only about ¼ of the bit combinations of a given byte are likely to occur.

Despite this criticism, the U.S. government insisted throughout the mid-1990s that 56-bit DES was secure and virtually unbreakable if appropriate precautions were taken. In response, RSA Laboratories sponsored a series of cryptographic challenges to prove that DES was no longer appropriate for use.

DES Challenge I was launched in March 1997. It was completed in 84 days by R. Verser in a collaborative effort using thousands of computers on the Internet.

The first DES II challenge lasted 40 days in early 1998. This problem was solved by distributed.net, a worldwide distributed computing network using the spare CPU cycles of computers around the Internet (participants in distributed.net's activities load a client program that runs in the background, conceptually similar to the SETI @Home "Search for Extraterrestrial Intelligence" project). The distributed.net systems were checking 28 *billion* keys per second by the end of the project.

The second DES II challenge lasted less than 3 days. On July 17, 1998, the Electronic Frontier Foundation (EFF) announced the construction of hardware that could brute-force a DES key in an average of 4.5 days. Called Deep Crack, the device could check 90 billion keys per second and cost only about $220,000 including design (it was erroneously and widely reported that subsequent devices could be built for as little as $50,000). Since the design is scalable, this suggests that an organization could build a DES cracker that could break 56-bit keys in an average of a day for as little as $1,000,000. Information about the hardware design and all software can be obtained from the EFF.

The DES III challenge, launched in January 1999, was broken is less than a day by the combined efforts of Deep Crack and distributed.net. This is widely considered to have been the final nail in DES's coffin.

The Deep Crack algorithm is actually quite interesting. The general approach that the DES Cracker Project took was not

to break the algorithm mathematically but instead to launch a brute-force attack by guessing every possible key. A 56-bit key yields 2^{56}, or about 72 quadrillion, possible values. So the DES cracker team looked for any shortcuts they could find! First, they assumed that *some* recognizable plaintext would appear in the decrypted string even though they didn't have a specific known plaintext block. They then applied all 2^{56} possible key values to the 64-bit block (I don't mean to make this sound simple!). The system checked to see if the decrypted value of the block was "interesting," which they defined as bytes containing one of the alphanumeric characters, space, or some punctuation. Since the likelihood of a single byte being "interesting" is about ¼, then the likelihood of the entire 8-byte stream being "interesting" is about $¼^8$, or 1/65536 ($½^{16}$). This dropped the number of possible keys that might yield positive results to about 2^{40}, or about a trillion.

They then made the assumption that an "interesting" 8-byte block would be followed by another "interesting" block. So, if the first block of ciphertext decrypted to something interesting, they decrypted the next block; otherwise, they abandoned this key. Only if the second block was also "interesting" did they examine the key closer. Looking for 16 consecutive bytes that were "interesting" meant that only 2^{24}, or 16 million, keys needed to be examined further. This further examination was primarily to see if the text made any sense. Note that possible "interesting" blocks might be `1hJ5&aB7` or `DEPOSITS`; the latter is more likely to produce a better result. And even a slow laptop today can search through lists of only a few million items in a relatively short period of time. (Interested readers are urged to read *Cracking DES* and EFF's Cracking DES page.)

It is well beyond the scope of this paper to discuss other forms of breaking DES and other codes. Nevertheless, it is worth mentioning a couple of forms of cryptanalysis that have been shown to be effective against DES. *Differential cryptanalysis*, invented in 1990 by E. Biham and A. Shamir (of RSA fame), is a chosen-plaintext attack. By selecting pairs of plaintext with particular differences, the cryptanalyst examines the differences in the resultant ciphertext pairs. *Linear plaintext*, invented by M. Matsui, uses a linear approximation to analyze the actions of a block cipher (including DES). Both of these attacks can be more efficient than brute force.

DES Variants

Once DES was "officially" broken, several variants appeared. But none of them came overnight; work at hardening DES had already been underway. In the early 1990s, there was a proposal to increase the security of DES by effectively increasing the key length by using multiple keys with multiple passes. But for this scheme to work, it had to first be shown that the DES function is **not** a *group*, as defined in mathematics. If DES was a group, then we could show that for two DES keys, X1 and X2, applied to some plaintext (P), we can find a single equivalent key, X3, that would provide the same result; i.e.,:

$$E_{X2}(E_{X1}(P)) = E_{X3}(P)$$

where $E_X(P)$ represents DES encryption of some plaintext *P* using DES key *X*. If DES were a group, it wouldn't matter how many keys and passes we applied to some plaintext; we could always find a single 56-bit key that would provide the same result.

As it happens, DES was proven to not be a group so that as we apply additional keys and passes, the effective key length increases. One obvious choice, then, might be to use two keys and two passes, yielding an effective key length of 112 bits. Let's call this Double-DES. The two keys, Y1 and Y2, might be applied as follows:

$$C = E_{Y2}(E_{Y1}(P))$$
$$P = D_{Y1}(D_{Y2}(C))$$

where $E_Y(P)$ and $D_Y(C)$ represent DES encryption and decryption, respectively, of some plaintext *P* and ciphertext *C*, respectively, using DES key *Y*.

So far, so good. But there's an interesting attack that can be launched against this "Double-DES" scheme. First, notice that the applications of the formula above can be thought of with the following individual steps (where C' and P' are intermediate results):

$$C' = E_{Y1}(P) \text{ and } C = E_{Y2}(C')$$
$$P' = D_{Y2}(C) \text{ and } P = D_{Y1}(P')$$

Unfortunately, C'=P'. That leaves us vulnerable to a simple *known plaintext* attack (sometimes called "Meet-in-the-middle") where the attacker knows some plaintext (P) and its matching ciphertext (C). To obtain C', the attacker needs to try all 2^{56} possible values of Y1 applied to P; to obtain P', the attacker needs to try all 2^{56} possible values of Y2 applied to C. Since C'=P', the attacker knows when a match has been achieved — after only $2^{56} + 2^{56} = 2^{57}$ key searches, only twice the work of brute-forcing DES. So "Double-DES" won't work.

Triple-DES (3DES), based upon the Triple Data Encryption Algorithm (TDEA), is described in FIPS 46-3. 3DES, which is not susceptible to a meet-in-the-middle attack, employs three DES passes and one, two, or three keys called K1, K2, and K3. Generation of the ciphertext (C) from a block of plaintext (P) is accomplished by:

$$C = E_{K3}(D_{K2}(E_{K1}(P)))$$

where $E_K(P)$ and $D_K(P)$ represent DES encryption and decryption, respectively, of some plaintext *P* using DES key *K*. (For obvious reasons, this is sometimes referred to as an *encrypt-decrypt-encrypt mode* operation.)

Decryption of the ciphertext into plaintext is accomplished by:

$$P = D_{K1}(E_{K2}(D_{K3}(C)))$$

The use of three, independent 56-bit keys provides 3DES with an effective key length of 168 bits. The specification also defines use of two keys where, in the operations above, K3 = K1; this provides an effective key length of 112 bits. Finally, a third keying option is to use a single key, so that K3 = K2 = K1 (in this case, the effective key length is 56 bits and 3DES applied to some plaintext, P, will yield the same ciphertext, C, as normal DES would with that same key). Given the relatively low cost of key storage and the modest increase in processing due to the use of longer keys, the best recommended practices are that 3DES be employed with three keys.

Another variant of DES, called DESX, is due to Ron Rivest. Developed in 1996, DESX is a very simple algorithm that greatly increases DES's resistance to brute-force attacks without increasing its computational complexity. In DESX, the plaintext input is XORed with 64 additional key bits prior to encryption and the output is likewise XORed with the 64 key bits. By adding just two XOR operations, DESX has an effective keylength of 120 bits against an exhaustive key-search attack. As it happens, DESX is no more immune to other types of more sophisticated attacks, such as differential or linear cryptanalysis, but brute-force is the primary attack vector on DES.

Closing Comments

Although DES has been deprecated and replaced by the Advanced Encryption Standard (AES) because of its vulnerability to a modestly-priced brute-force attack, many applications continue to rely on DES for security, and many software designers and implementers continue to include DES in new applications. In some cases, use of DES is wholly appropriate but, in general, DES should not continue to be promulgated in production software and hardware. RFC 4772 discusses the security implications of employing DES.

On a final note, readers may be interested in seeing The Illustrated DES Spreadsheet (J. Hughes, 2004), an Excel implementation of DES, or J.O. Grabbe's The DES Algorithm Illustrated.

5.5. Pretty Good Privacy (PGP)

Pretty Good Privacy (PGP) is one of today's most widely used public key cryptography programs. Developed by Philip Zimmermann in the early 1990s and long the subject of controversy, PGP is available as a plug-in for many e-mail clients, such as Claris Emailer, Microsoft Outlook/Outlook Express, and Qualcomm Eudora.

PGP can be used to sign or encrypt e-mail messages with the mere click of the mouse. Depending upon the version of PGP, the software uses SHA or MD5 for calculating the message hash; CAST, Triple-DES, or IDEA for encryption; and RSA or DSS/Diffie-Hellman for key exchange and digital signatures.

When PGP is first installed, the user has to create a key-pair. One key, the public key, can be advertised and widely circulated. The private key is protected by use of a *passphrase*. The passphrase has to be entered every time the user accesses their private key.

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

Hi Carol.

What was that pithy Groucho Marx quote?

/kess

-----BEGIN PGP SIGNATURE-----
Version: PGP for Personal Privacy 5.0
Charset: noconv

iQA/AwUBNFUdO5WOcz5SFtuEEQJx/ACaAgR97+vvDU6XWELV/GANjAAgBtUAnjG3
Sdfw2JgmZIOLNjFe7jP0Y8/M
=jUAU
-----END PGP SIGNATURE-----
```

FIGURE 7: A PGP signed message. The sender uses their private key; at the destination, the sender's e-mail address yields the public key from the receiver's keyring.

Figure 7 shows a PGP signed message. This message will not be kept secret from an eavesdropper, but a recipient can be assured that the message has not been altered from what the sender transmitted. In this instance, the sender signs the message using their own private key. The receiver uses the sender's public key to verify the signature; the public key is taken from the receiver's keyring based on the sender's e-mail address. Note that the signature process does not work unless the sender's public key is on the receiver's keyring.

```
-----BEGIN PGP MESSAGE-----
Version: PGP for Personal Privacy 5.0
MessageID: DAdVB3wzpBr3YRunZwYvhK5gBKBXOb/m

qANQR1DBwU4D/TlT68XXuiUQCADfj2o4b4aFYBcWumA7hR1Wvz9rbv2BR6WbEUsy
ZBIEFtjyqCd96qF38sp9IQiJIKlNaZfx2GLRWikPZwchUXxB+AA5+lqsG/ELBvRa
c9XefaYpbbAZ6z6LkOQ+eE0XASe7aEEPfdxvZZT37dVyiyxuBBRYNLN8Bphdr2zv
z/9Ak4/OLnLiJRk05/2UNE5Z0a+3lcvITMmfGajvRhkXqocavPOKiin3hv7+Vx88
uLLem2/fQHZhGcQvkqZVqXx8SmNw5gzuvwjV1WHj9muDGBY0MkjiZIRI7azWnoU9
3KCnmpR60VO4rDRAS5uGl9fioSvze+q8XqxubaNsgdKkoD+tB/4u4c4tznLfw1L2
YBS+dzFDw5desMFSo7JkecAS4NB9jAu9K+f7PTAsesCBNETDd49BTOFFTWWavAfE
gLYcPrcn4s3EriUgvL3OzPR4P1chNu6sa3ZJkTBbriDoA3VpnqG3hxqfNyOlqAka

mJJuQ53Ob9ThaFH8YcE/VqUFdw+bQtrAJ6NpjIxi/x0FfOInhC/bBw7pDLXBFNaX
HdlLQRPQdrmnWskKznOSarxq4GjpRTQo4hpCRJJ5aU7tZO9HPTZXFG6iRIT0wa47


AR5nvkEKoIAjW5HaDKiJriuWLdtN4OXecWvxFsjR32ebz76U8aLpAK87GZEyTzBx
dV+lH0hwyT/y1cZQ/E5USePP4oKWF4uqquPee1OPeFMBo4CvuGyhZXD/18Ft/53Y
WIebvdiCqsOoabK3jEfdGExce63zDI0=
=MpRf
-----END PGP MESSAGE-----
```

FIGURE 8: A PGP encrypted message. The receiver's e-mail address is the pointer to the public key in the sender's keyring. At the destination side, the receiver uses their own private key.

Figure 8 shows a PGP encrypted message (PGP compresses the file, where practical, prior to encryption because encrypted files have a high degree of randomness and, therefore, cannot be efficiently compressed). In this example, public key methods are used to exchange the session key for the actual message encryption that employs secret-key cryptography. In this case, the receiver's e-mail address is the pointer to the public key in the sender's keyring; in fact, the same message can be sent to multiple recipients and the message will not be significantly longer since all that needs to be added is the session key encrypted by each receiver's public key. When the message is received, the recipient will use their private key to extract the session secret key to successfully decrypt the message (Figure 9).

```
Hi Gary,

"Outside of a dog, a book is man's best friend.
Inside of a dog, it's too dark to read."

Carol
```

FIGURE 9: The decrypted message.

It is worth noting that PGP was one of the first so-called "hybrid cryptosystems" that combined aspects of SKC and PKC. When Zimmermann was first designing PGP in the late-1980s, he wanted to use RSA to encrypt the entire message. The PCs of the days, however, suffered significant performance degradation when executing RSA so he hit upon the idea of using SKC to encrypt the message and PKC to encrypt the SKC key.

PGP went into a state of flux in 2002. Zimmermann sold PGP to Network Associates, Inc. (NAI) in 1997 and himself resigned from NAI in early 2001. In March 2002, NAI announced that they were dropping support for the commercial version of PGP having failed to find a buyer for the product willing to pay what NAI wanted. In August 2002, PGP was purchased from NAI by PGP Corp. (*http://www.pgp.com/*). Meanwhile, there are many freeware versions of PGP available through the International PGP Page and the OpenPGP Alliance. Also check out the GNU Privacy Guard (GnuPG), a GNU project implementation of OpenPGP (defined in RFC 2440).

5.6. IP Security (IPsec) Protocol

> **NOTE:** The information in this section assumes that the reader is familiar with the Internet Protocol (IP), at least to the extent of the packet format and header contents. More information about IP can be found in *An Overview of TCP/IP Protocols and the Internet*. More information about IPv6 can be found in IPv6: The Next Generation Internet Protocol.

The Internet and the TCP/IP protocol suite were not built with security in mind. This statement is not meant as a criticism; the baseline UDP, TCP, IP, and ICMP protocols were written in 1980 and built for the relatively closed ARPANET community. TCP/IP wasn't designed for the commercial-grade financial transactions that they now see nor for virtual private networks (VPNs) on the Internet. To bring TCP/IP up to today's security necessities, the Internet Engineering Task Force (IETF) formed the IP Security Protocol Working Group which, in turn, developed the IP Security (IPsec) protocol. IPsec is not a single protocol, in fact, but a suite of protocols providing a mechanism to provide data integrity, authentication, privacy, and nonrepudiation for the classic Internet Protocol (IP). Although intended primarily for IP version 6 (IPv6), IPsec can also be employed by the current version of IP, namely IP version 4 (IPv4).

As shown in Table 3, IPsec is described in nearly a dozen RFCs. RFC 4301, in particular, describes the overall IP security architecture and RFC 2411 provides an overview of the IPsec protocol suite and the documents describing it.

IPsec can provide either message authentication and/or encryption. The latter requires more processing than the former, but will probably end up being the preferred usage for applications such as VPNs and secure electronic commerce.

Central to IPsec is the concept of a *security association (SA)*. Authentication and confidentiality using AH or ESP use SAs and a primary role of IPsec key exchange it to establish and maintain SAs. An SA is a simplex (one-way or unidirectional) logical connection between two communicating IP endpoints that provides security services to the traffic carried by it using either AH or ESP procedures. The endpoint of an SA can be an IP host or IP security gateway (e.g., a proxy server, VPN server, etc.). Providing security to the more typical scenario of two-way (bi-directional) communication between two endpoints requires the establishment of two SAs (one in each direction).

An SA is uniquely identified by a 3-tuple composed of:

- Security Parameter Index (SPI), a 32-bit identifier of the connection
- IP Destination Address
- security protocol (AH or ESP) identifier

The IP Authentication Header (AH), described in RFC 4302, provides a mechanism for data integrity and data origin authentication for IP packets using HMAC with MD5 (RFC 2403), HMAC with SHA-1 (RFC 2404), or HMAC with RIPEMD (RFC 2857). See also RFC 4305.

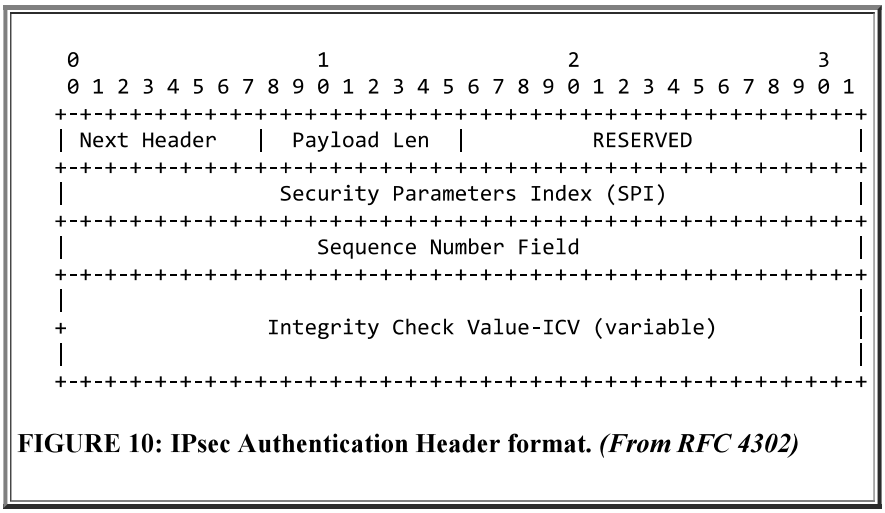


FIGURE 10: IPsec Authentication Header format. ***(From RFC 4302)***

Figure 10 shows the format of the IPsec AH. The AH is merely an additional header in a packet, more or less representing another protocol layer above IP (this is shown in Figure 12 below). Use of the IP AH is indicated by placing the value 51

(0x33) in the IPv4 Protocol or IPv6 Next Header field in the IP packet header. The AH follows mandatory IPv4/IPv6 header fields and precedes higher layer protocol (e.g., TCP, UDP) information. The contents of the AH are:

- *Next Header:* An 8-bit field that identifies the type of the next payload after the Authentication Header.
- *Payload Length:* An 8-bit field that indicates the length of AH in 32-bit words (4-byte blocks), minus "2". [The rationale for this is somewhat counter intuitive but technically important. All IPv6 extension headers encode the header extension length (Hdr Ext Len) field by first subtracting 1 from the header length, which is measured in 64-bit words. Since AH was originally developed for IPv6, it is an IPv6 extension header. Since its length is measured in 32-bit words, however, the Payload Length is calculated by subtracting 2 (32 bit words) to maintain consistency with IPv6 coding rules.] In the default case, the three 32-bit word fixed portion of the AH is followed by a 96-bit authentication value, so the Payload Length field value would be 4.
- *Reserved:* This 16-bit field is reserved for future use and always filled with zeros.
- *Security Parameters Index (SPI):* An arbitrary 32-bit value that, in combination with the destination IP address and security protocol, uniquely identifies the Security Association for this datagram. The value 0 is reserved for local, implementation-specific uses and values between 1-255 are reserved by the Internet Assigned Numbers Authority (IANA) for future use.
- *Sequence Number:* A 32-bit field containing a sequence number for each datagram; initially set to 0 at the establishment of an SA. AH uses sequence numbers as an anti-replay mechanism, to prevent a "person-in-the-middle" attack. If anti-replay is enabled (the default), the transmitted Sequence Number is never allowed to cycle back to 0; therefore, the sequence number must be reset to 0 by establishing a new SA prior to the transmission of the 2^{32}nd packet.
- *Authentication Data:* A variable-length, 32-bit aligned field containing the Integrity Check Value (ICV) for this packet (default length = 96 bits). The ICV is computed using the authentication algorithm specified by the SA, such as DES, MD5, or SHA-1. Other algorithms *may* also be supported.

The IP Encapsulating Security Payload (ESP), described in RFC 4303, provides message integrity and privacy mechanisms in addition to authentication. As in AH, ESP uses HMAC with MD5, SHA-1, or RIPEMD authentication (RFC 2403/RFC 2404/RFC 2857); privacy is provided using DES-CBC encryption (RFC 2405), NULL encryption (RFC 2410), other CBC-mode algorithms (RFC 2451), or AES (RFC 3686). See also RFC 4305 and RFC 4308.

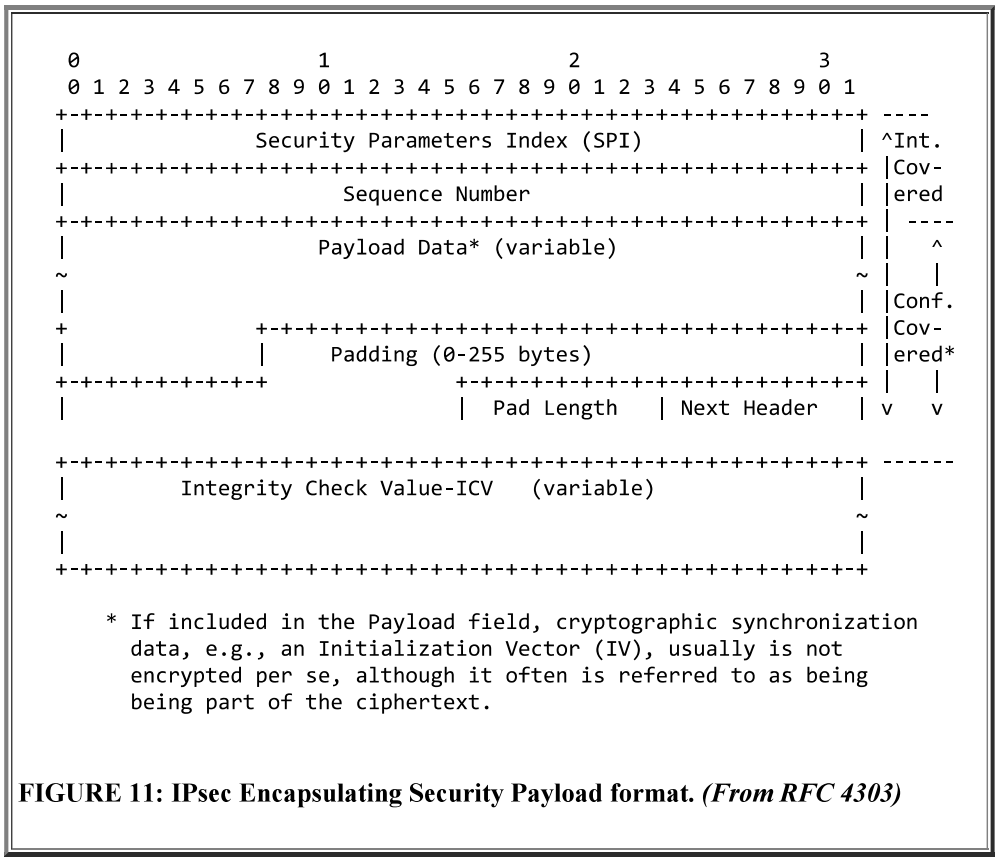


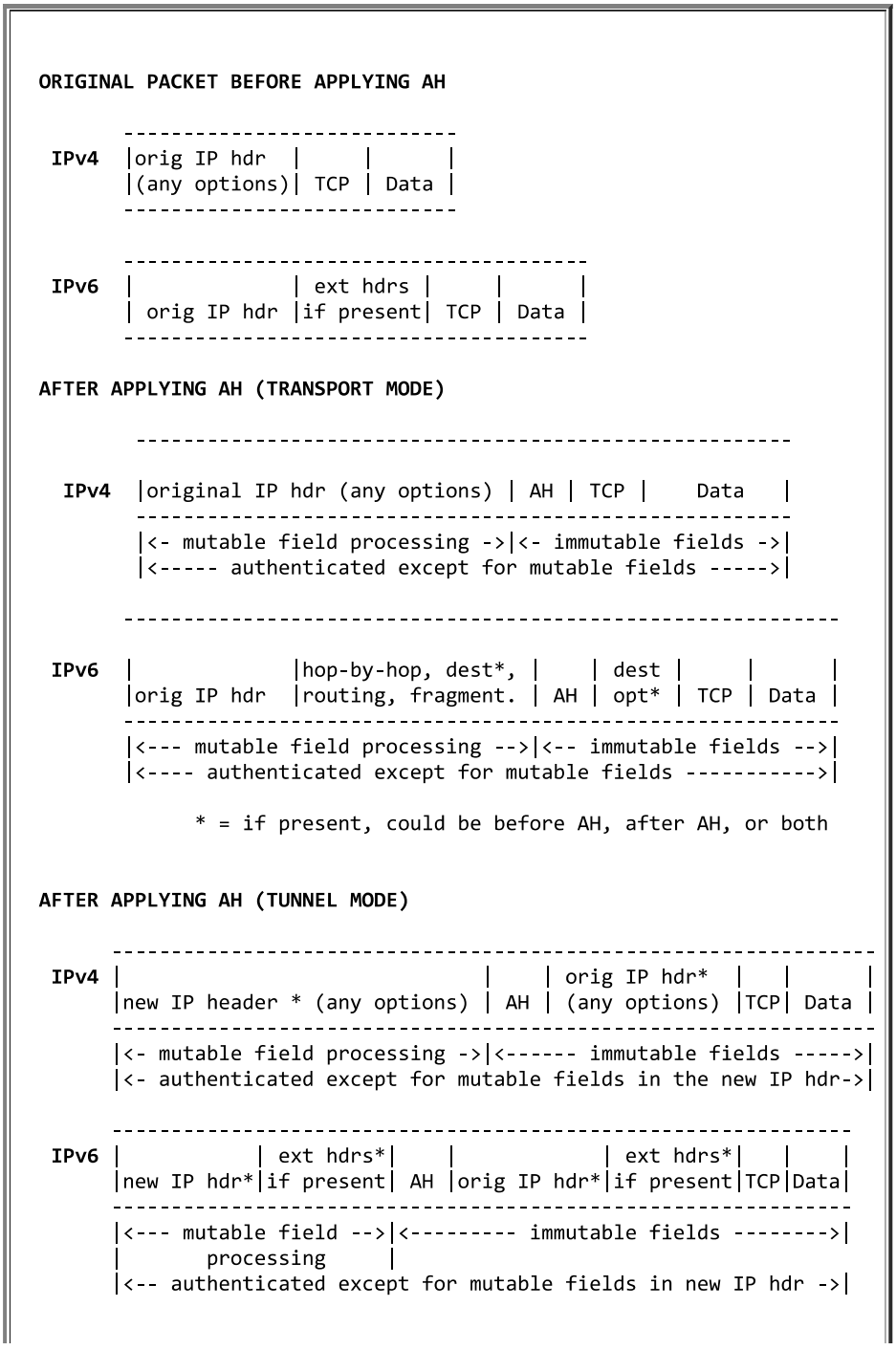
FIGURE 11: IPsec Encapsulating Security Payload format. ***(From RFC 4303)***

Figure 11 shows the format of the IPsec ESP information. Use of the IP ESP format is indicated by placing the value 50 (0x32) in the IPv4 Protocol or IPv6 Next Header field in the IP packet header. The ESP header (i.e., SPI and sequence number) follows mandatory IPv4/IPv6 header fields and precedes higher layer protocol (e.g., TCP, UDP) information. The contents of the ESP packet are:

- *Security Parameters Index:* (see description for this field in the AH, above.)

- *Sequence Number:* (see description for this field in the AH, above.)
- *Payload Data:* A variable-length field containing data as described by the Next Header field. The contents of this field could be encrypted higher layer data or an encrypted IP packet.
- *Padding:* Between 0 and 255 octets of padding may be added to the ESP packet. There are several applications that might use the padding field. First, the encryption algorithm that is used may require that the plaintext be a multiple of some number of bytes, such as the block size of a block cipher; in this case, the Padding field is used to fill the plaintext to the size required by the algorithm. Second, padding may be required to ensure that the ESP packet and resulting ciphertext terminate on a 4-byte boundary. Third, padding may be used to conceal the actual length of the payload. Unless another value is specified by the encryption algorithm, the Padding octets take on the value 1, 2, 3, ... starting with the first Padding octet. This scheme is used because, in addition to being simple to implement, it provides some protection against certain forms of "cut and paste" attacks.
- *Pad Length:* An 8-bit field indicating the number of bytes in the Padding field; contains a value between 0-255.
- *Next Header:* An 8-bit field that identifies the type of data in the Payload Data field, such as an IPv6 extension header or a higher layer protocol identifier.
- *Authentication Data:* (see description for this field in the AH, above.)

Two types of SAs are defined in IPsec, regardless of whether AH or ESP is employed. A *transport mode SA* is a security association between two hosts. Transport mode provides the authentication and/or encryption service to the higher layer protocol. This mode of operation is only supported by IPsec hosts. A *tunnel mode SA* is a security association applied to an IP tunnel. In this mode, there is an "outer" IP header that specifies the IPsec destination and an "inner" IP header that specifies the destination for the IP packet. This mode of operation is supported by both hosts and security gateways.

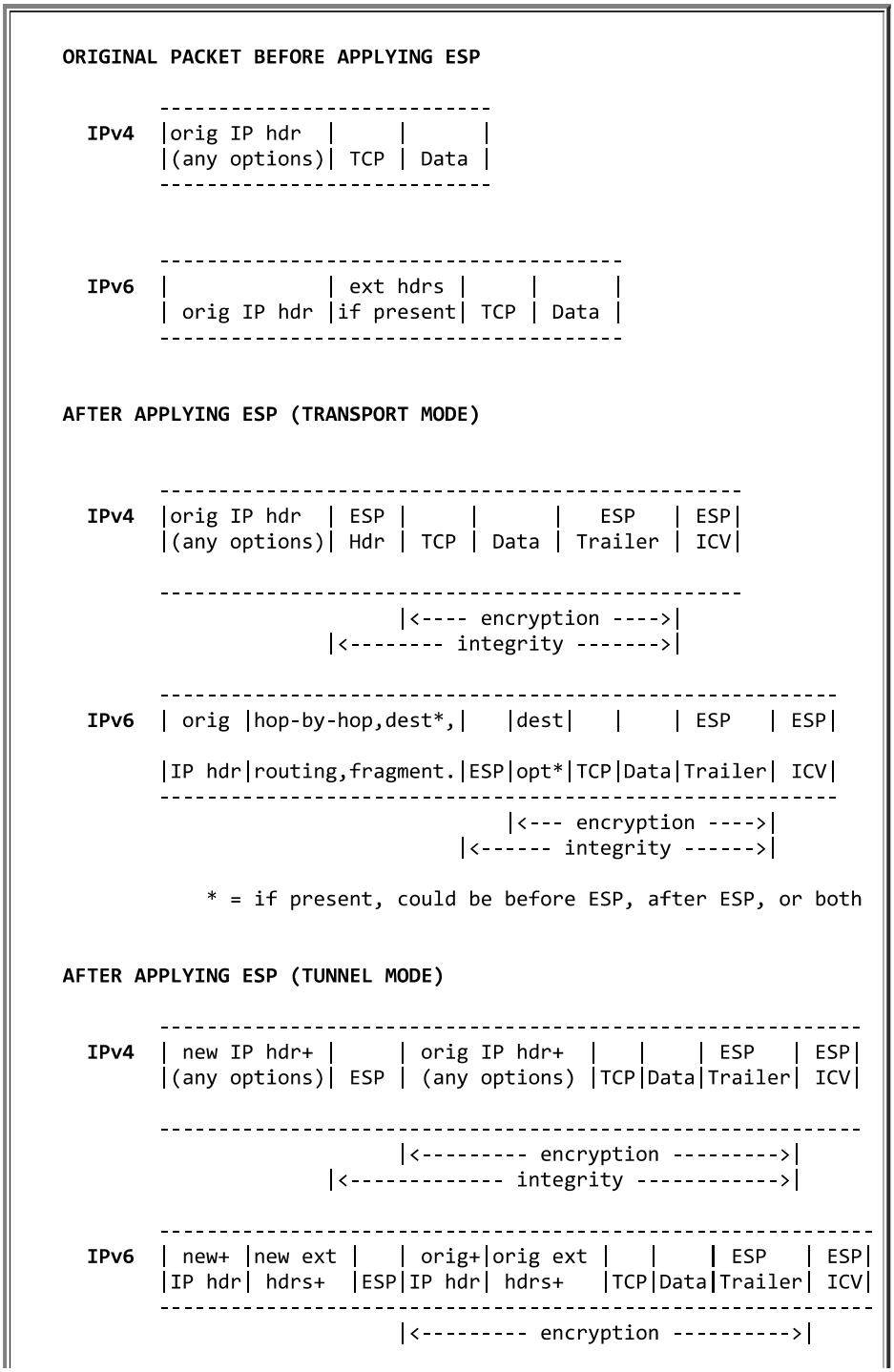


```
        * = if present, construction of outer IP hdr/extensions and
            modification of inner IP hdr/extensions is discussed in
            the Security Architecture document.
```

FIGURE 12: IPsec tunnel and transport modes for AH. ***(Adapted from RFC 4302)***

Figure 12 show the IPv4 and IPv6 packet formats when using AH in both transport and tunnel modes. Initially, an IPv4 packet contains a normal IPv4 header (which may contain IP options), followed by the higher layer protocol header (e.g., TCP or UDP), followed by the higher layer data itself. An IPv6 packet is similar except that the packet starts with the mandatory IPv6 header followed by any IPv6 extension headers, and then followed by the higher layer data.

Note that in both transport and tunnel modes, the **entire** IP packet is covered by the authentication *except for the mutable fields.* A field is *mutable* if its value might change during transit in the network; IPv4 mutable fields include the fragment offset, time to live, and checksum fields. Note, in particular, that the address fields are *not* mutable.



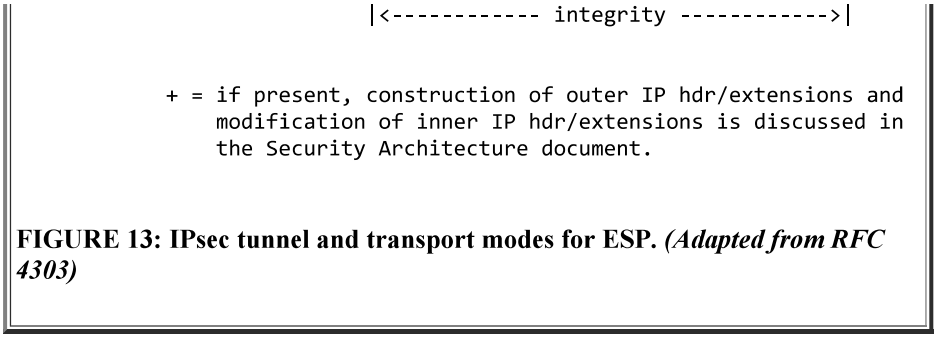


FIGURE 13: IPsec tunnel and transport modes for ESP. ***(Adapted from RFC 4303)***

Figure 13 shows the IPv4 and IPv6 packet formats when using ESP in both transport and tunnel modes.

- As with AH, we start with a standard IPv4 or IPv6 packet.
- In transport mode, the higher layer header and data, as well as ESP trailer information, is encrypted and the entire ESP packet is authenticated. In the case of IPv6, some of the IPv6 extension options can precede or follow the ESP header.
- In tunnel mode, the original IP packet is encrypted and placed inside of an "outer" IP packet, while the entire ESP packet is authenticated.

Note a significant difference in the scope of ESP and AH. AH authenticates the entire packet transmitted on the network whereas ESP only covers a portion of the packet transmitted on the network (the higher layer data in transport mode and the entire original packet in tunnel mode). The reason for this is straight-forward; in AH, the authentication data for the transmission fits neatly into an additional header whereas ESP creates an entirely new packet which is the one encrypted and/or authenticated. But the ramifications are significant. ESP transport mode as well as AH in both modes protect the IP address fields of the original transmissions. Thus, using IPsec in conjunction with network address translation (NAT) *might* be problematic because NAT changes the values of these fields *after* IPsec processing.

The third component of IPsec is the establishment of security associations and key management. These tasks can be accomplished in one of two ways.

The simplest form of SA and key management is manual management. In this method, a security administer or other individual manually configures each system with the key and SA management data necessary for secure communication with other systems. Manual techniques are practical for small, reasonably static environments but they do not scale well.

For successful deployment of IPsec, however, a scalable, automated SA/key management scheme is necessary. Several protocols have defined for these functions:

- The Internet Security Association and Key Management Protocol (ISAKMP) defines procedures and packet formats to establish, negotiate, modify and delete security associations, and provides the framework for exchanging information about authentication and key management (RFC 2407/RFC 2408). ISAKMP's security association and key management is totally separate from key exchange.
- The OAKLEY Key Determination Protocol (RFC 2412) describes a scheme by which two authenticated parties can exchange key information. OAKLEY uses the Diffie-Hellman key exchange algorithm.
- The Internet Key Exchange (IKE) algorithm (RFC 2409) is the default automated key management protocol for IPsec.
- An alternative to IKE is Photuris (RFC 2522/RFC 2523), a scheme for establishing short-lived session-keys between two authenticated parties without passing the session-keys across the Internet. IKE typically creates keys that may have very long lifetimes.

On a final note, IPsec authentication for both AH and ESP uses a scheme called *HMAC*, a keyed-hashing message authentication code described in FIPS 198 and RFC 2104. HMAC uses a shared secret key between two parties rather than public key methods for message authentication. The generic HMAC procedure can be used with just about any hash algorithm, although IPsec specifies support for at least MD5 and SHA-1 because of their widespread use.

In HMAC, both parties share a secret key. The secret key will be employed with the hash algorithm in a way that provides mutual authentication without transmitting the key on the line. IPsec key management procedures will be used to manage key exchange between the two parties.

Recall that hash functions operate on a fixed-size block of input at one time; MD5 and SHA-1, for example, work on 64 byte blocks. These functions then generate a fixed-size hash value; MD5 and SHA-1, in particular, produce 16 byte (128 bit) and 20 byte (160 bit) output strings, respectively. For use with HMAC, the secret key (K) should be at least as long as the hash output.

The following steps provide a simplified, although reasonably accurate, description of how the HMAC scheme would work with a particular plaintext MESSAGE:

1. Alice pads K so that it is as long as an input block; call this padded key K_p. Alice computes the hash of the padded key followed by the message, i.e., HASH (K_p:MESSAGE).
2. Alice transmits MESSAGE and the hash value.
3. Bob has also padded K to create K_p. He computes HASH (K_p:MESSAGE) on the incoming message.
4. Bob compares the computed hash value with the received hash value. If they match, then the sender — Alice — must know the secret key and the message is authenticated.

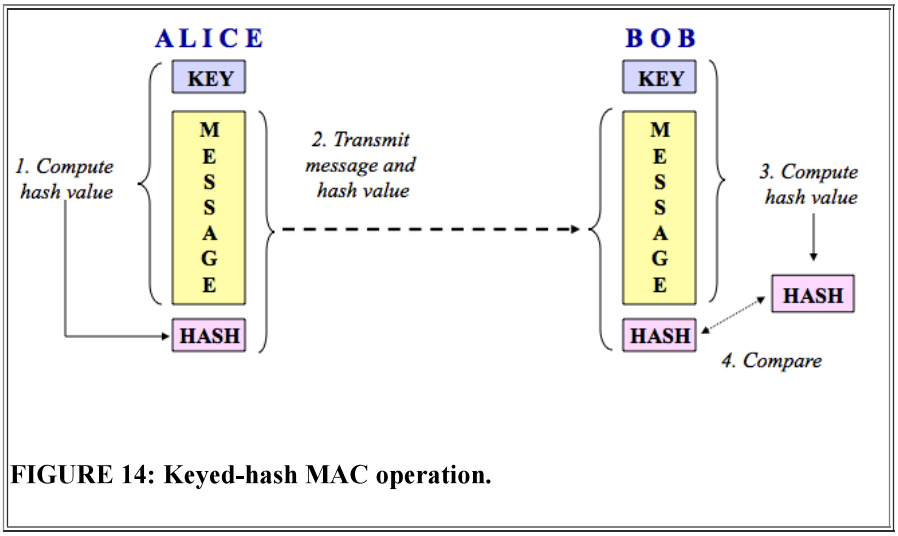


FIGURE 14: Keyed-hash MAC operation.

5.7. The SSL Family of Secure Transaction Protocols for the World Wide Web

The Secure Sockets Layer (SSL) protocol was developed by Netscape Communications to provide application-independent secure communication over the Internet for protocols such as the Hypertext Transfer Protocol (HTTP). SSL employs RSA and X.509 certificates during an initial handshake used to authenticate the server (client authentication is optional). The client and server then agree upon an encryption scheme. SSL v2.0 (1995), the first version publicly released, supported RC2 and RC4 with 40-bit keys. SSL v3.0 (1996) added support for DES, RC4 with a 128-bit key, and 3DES with a 168-bit key, all along with either MD5 or SHA-1 message hashes; this protocol is described in RFC 6101.

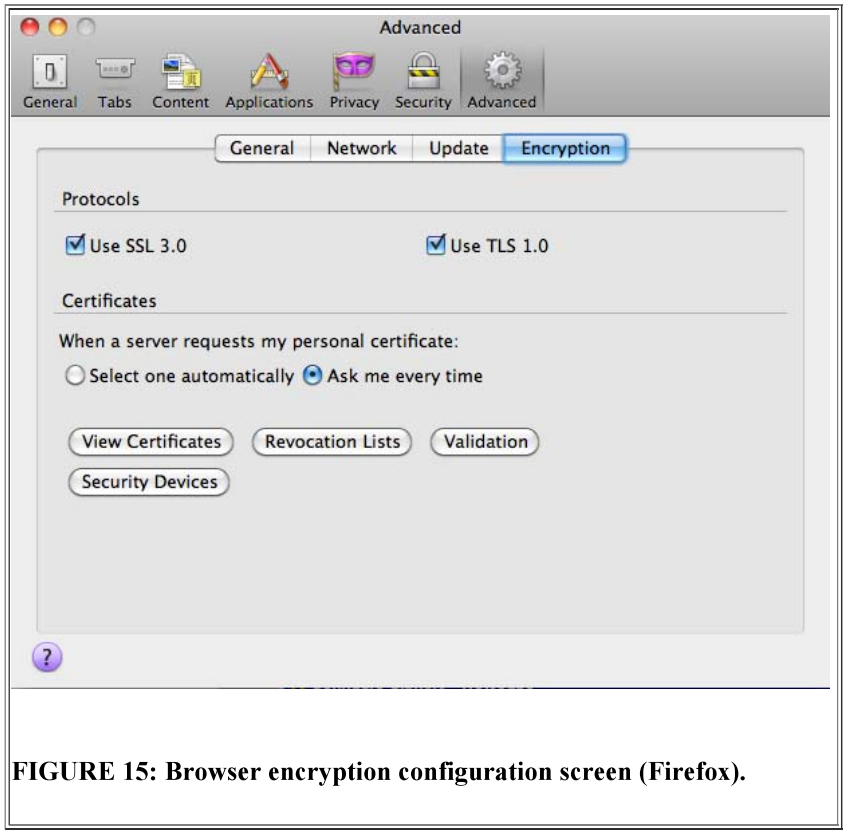


FIGURE 15: Browser encryption configuration screen (Firefox).

In 1997, SSL v3 was found to be breakable. By this time, the Internet Engineering Task Force (IETF) had already started

work on a new, non-proprietary protocol called Transport Layer Security (TLS), described in RFC 2246 (1999). TLS extends SSL and supports additional crypto schemes, such as Diffie-Hellman key exchange and DSS digital signatures; RFC 4279 describes the pre-shared key crypto schemes supported by TLS. TLS is backward compatible with SSL (and, in fact, is recognized as SSL v3.1). SSL v3.0 and TLS v1.0 are the commonly supported versions on servers and browsers today (Figure 15); SSL v2.0 is rarely found today and, in fact, RFC 6176-compliant client and servers that support TLS will never negotiate the use of SSL v2.

In 2002, a cipher block chaining (CBC) vulnerability was described for TLS v1.0. In 2011, the theoretical became practical when a CBC proof-of-concept exploit was released. Meanwhile, TLS v1.1 was defined in 2006 (RFC 4346), adding protection against v1.0's CBC vulnerability. In 2008, TLS v1.2 was defined (RFC 5246), adding several additional cryptographic options. Today, users are urged to use TLS v1.2 or v1.1 in lieu of any earlier versions.

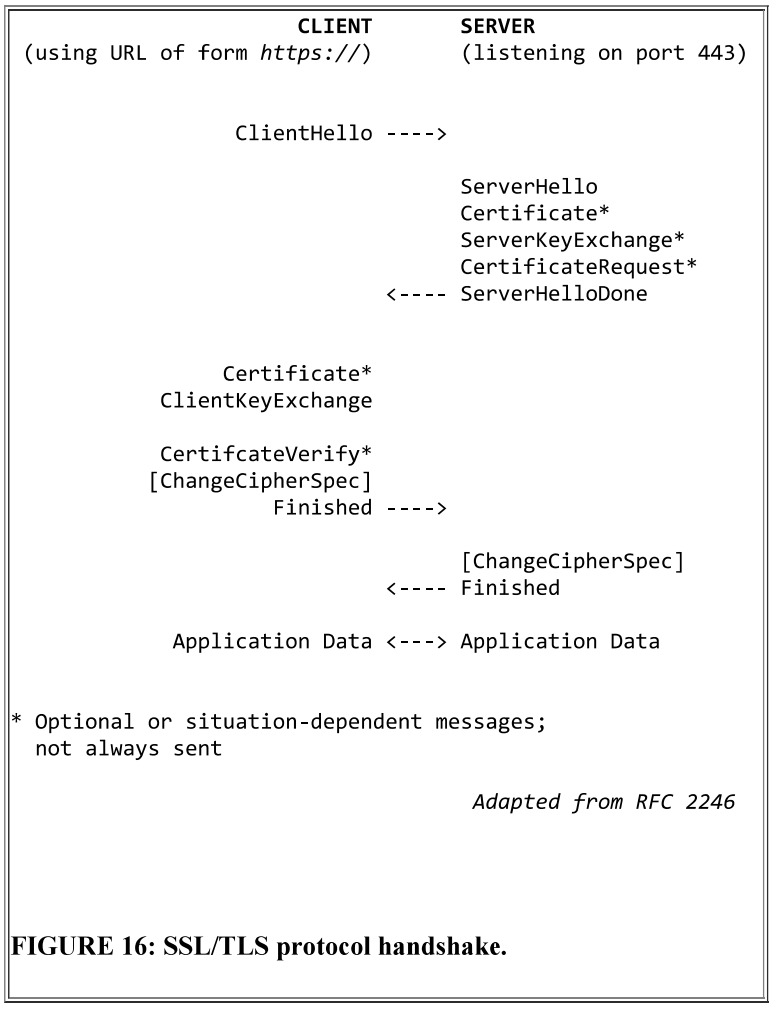


FIGURE 16: SSL/TLS protocol handshake.

Figure 16 shows the basic TLS (and SSL) message exchanges:

1. URLs specifying the protocol *https://* are directed to HTTP servers secured using SSL/TLS. The client will automatically try to make a TCP connection to the server at port 443. The client initiates the secure connection by sending a `ClientHello` message containing a Session identifier, highest SSL version number supported by the client, and lists of supported crypto and compression schemes (in preference order).
2. The server examines the Session ID and if it is still in the server's cache, it will attempt to re-establish a previous session with this client. If the Session ID is not recognized, the server will continue with the handshake to establish a secure session by responding with a `ServerHello` message. The `ServerHello` repeats the Session ID, indicates the SSL version to use for this connection (which will be the highest SSL version supported by the server and client), and specifies which encryption method and compression method to be used for this connection.
3. There are a number of other optional messages that the server might send, including:
   - `Certificate`, which carries the server's X.509 public key certificate (and, generally, the server's public key). This message will always be sent unless the client and server have already agreed upon some form of anonymous key exchange. (This message is normally sent.)
   - `ServerKeyExchange`, which will carry a premaster secret when the server's `Certificate` message does not contain enough data for this purpose; used in some key exchange schemes.
   - `CertificateRequest`, used to request the client's certificate in those scenarios where client authentication is performed.
   - `ServerHelloDone`, indicating that the server has completed its portion of the key exchange handshake.

4. The client now responds with a series of mandatory and optional messages:
   - `Certificate`, contains the client's public key certificate when it has been requested by the server.
   - `ClientKeyExchange`, which usually carries the secret key to be used with the secret key crypto scheme.
   - `CertificateVerify`, used to provide explicit verification of a client's certificate if the server is authenticating the client.
5. TLS includes the change cipher spec protocol to indicate changes in the encryption method. This protocol contains a single message, `ChangeCipherSpec`, which is encrypted and compressed using the current (rather than the new) encryption and compression schemes. The `ChangeCipherSpec` message is sent by both client and server to notify the other station that all following information will employ the newly negotiated cipher spec and keys.
6. The `Finished` message is sent after a `ChangeCipherSpec` message to confirm that the key exchange and authentication processes were successful.
7. At this point, both client and server can exchange application data using the session encryption and compression schemes.

Side Note: It would probably be helpful to make some mention of SSL (or, more properly, TLS) as it is used today. Most of us have used SSL to engage in a secure, private transaction with some vendor. The steps are something like this. During the SSL exchange with the vendor's secure server, the server sends its certificate to our client software. The certificate includes the vendor's public key and a signature from the CA that issued the vendor's certificate. Our browser software is shipped with the major CAs' certificates which contains their public key; in that way we authenticate the server. Note that the server does *not* use a certificate to authenticate us! Instead, we are generally authenticated when we provide our credit card number; the server checks to see if the card purchase will be authorized by the credit card company and, if so, considers us valid and authenticated! While bidirectional authentication is certainly supported by SSL, this form of asymmetric authentication is more commonly employed today since most users don't have certificates.

Microsoft's Server Gated Cryptography (SGC) protocol is another, albeit now defunct, extension to SSL/TLS. For several decades, it has been illegal to generally export products from the U.S. that employed secret-key cryptography with keys longer than 40 bits. For that reason, SSL/TLS has an exportable version with weak (40-bit) keys and a domestic (North American) version with strong (128-bit) keys. Within the last several years, however, use of strong SKC has been approved for the worldwide financial community. SGC is an extension to SSL that allows financial institutions using Windows NT servers to employ strong cryptography. Both the client and server must implement SGC and the bank must have a valid SGC certificate. During the initial handshake, the server will indicate support of SGC and supply its SGC certificate; if the client wishes to use SGC and validates the server's SGC certificate, the session can employ 128-bit RC2, 128-bit RC4, 56-bit DES, or 168-bit 3DES. Microsoft supports SGC in the Windows 95/98/NT versions of Internet Explorer 4.0, Internet Information Server (IIS) 4.0, and Money 98.

As mentioned above, SSL was designed to provide application-independent transaction security for the Internet. Although the discussion above has focused on HTTP over SSL (https/TCP port 443), SSL is also applicable to:

Protocol	TCP Port Name/Number
File Transfer Protocol (FTP)	ftps-data/989 & ftps/990
Internet Message Access Protocol v4 (IMAP4)	imaps/993
Lightweight Directory Access Protocol (LDAP)	ldaps/636
Network News Transport Protocol (NNTP)	nntps/563
Post Office Protocol v3 (POP3)	pop3s/995
Telnet	telnets/992

TLS was originally designed to operate over TCP. The IETF developed the Datagram Transport Layer Security (DTLS) protocol, based upon TLS, to operate over UDP. DTLS v1.2 is described in RFC 6347. (DTLS v1.0 can be found in RFC 4347.) RFC 6655 describes a suite of AES in Counter with Cipher Block Chaining - Message Authentication Code (CBC-MAC) Mode (CCM) ciphers for use with TLS and DTLS. An interesting analysis of the TLS protocol can be found in the paper "Analysis and Processing of Cryptographic Protocols" by Cowie.

Finally, a vulnerability in the OpenSSL Library was discovered in 2014. Known as Heartbleed, this vulnerability had apparently been introduced into OpenSSL in late 2011 with the introduction of a feature called *heartbeat*. Heartbleed exploited an implementation flaw in order to exfiltrate keying material from an SSL server (or some SSL clients, in what is known at *reverse Heartbleed*); the flaw allowed an attacker to grab 64 KB blocks from RAM. Heartbleed is known to only affect OpenSSL v1.0.1 through v1.0.1f; the exploit was patched in v1.0.1g. In addition, the OpenSSL 0.9.8 and 1.0.0 families are not vulnerable. Note also that Heartbleed affects some versions of the Android operating system, notably v4.1.0 and v4.1.1 (and some, possibly custom, implementations of v4.2.2). *Note that Heartbleed did **not** exploit a flaw in the SSL protocol, but rather a flaw in the OpenSSL implementation.*

5.8. Elliptic Curve Cryptography (ECC)

In general, public-key cryptography systems use hard-to-solve problems as the basis of the algorithm. The most predominant algorithm today for public-key cryptography is RSA, based on the prime factors of very large integers.

While RSA can be successfully attacked, the mathematics of the algorithm have not been comprised, per se; instead, computational brute-force has broken the keys. The defense is "simple" — keep the size of the integer to be factored ahead of the computational curve!

In 1985, Elliptic Curve Cryptography (ECC) was proposed independently by cryptographers Victor Miller (IBM) and Neal Koblitz (University of Washington). ECC is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). Like the prime factorization problem, ECDLP is another "hard" problem that is deceptively simple to state: Given two points, P and Q, on an elliptic curve, find the integer *n*, if it exists, such that P = nQ.

Elliptic curves combine number theory and algebraic geometry. These curves can be defined over any field of numbers (i.e., real, integer, complex) although we generally see them used over finite fields for applications in cryptography. An elliptic curve consists of the set of real numbers (x,y) that satisfies the equation:

$$y^2 = x^3 + ax + b$$

The set of all of the solutions to the equation forms the elliptic curve. Changing *a* and *b* changes the shape of the curve, and small changes in these parameters can result in major changes in the set of (x,y) solutions.

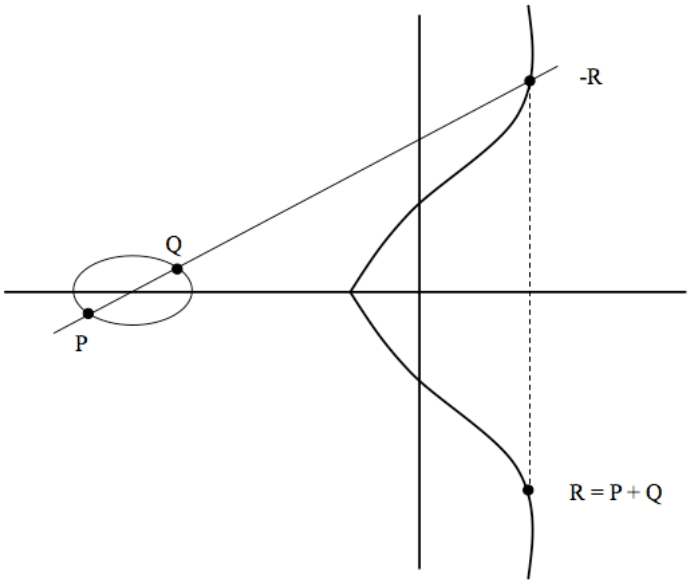


FIGURE 17: Elliptic curve addition.

Figure 17 shows the addition of two points on an elliptic curve. Elliptic curves have the interesting property that adding two points on the elliptic curve yields a third point on the curve. Therefore, adding two points, P and Q, gets us to point R, also on the curve. Small changes in P or Q can cause a large change in the position of R.

So let's go back to the original problem statement from above. The point Q is calculated as a multiple of the starting point, P, *or*, Q = nP. An attacker might know P and Q but finding the integer, *n*, is a difficult problem to solve. Q (i.e., nP) is the public key and *n* is the private key.

ECC may be employed with many Internet standards, including CCITT X.509 certificates and certificate revocation lists (CRLs), Internet Key Exchange (IKE), Transport Layer Security (TLS), XML signatures, and applications or protocols based on the cryptographic message syntax (CMS). RFC 5639 proposes a set of elliptic curve domain parameters over finite prime fields for use in these cryptographic applications and RFC 6637 proposes additional elliptic curves for use with OpenPGP.

RSA had been the mainstay of PKC for over a quarter-century. ECC, however, is emerging as a replacement in some environments because it provides similar levels of security compared to RSA but with significantly reduced key sizes. NIST use the following table to demonstrate the key size relationship between ECC and RSA, and the appropriate choice of AES key size:

TABLE 4. ECC and RSA Key Comparison.

ECC Key Size	RSA Key Size	Key-Size	AES Key Size

		Ratio	
163	1,024	1:6	n/a
256	3,072	1:12	128
384	7,680	1:20	192
512	15,360	1:30	256
Key sizes in bits.		*Source: Certicom, NIST*	

Since the ECC key sizes are so much shorter than comparable RSA keys, the length of the public key and private key is much shorter in elliptic curve cryptosystems. This results into faster processing times, and lower demands on memory and bandwidth; some studies have found that ECC is faster than RSA for signing and decryption, but slower for signature verification and encryption.

ECC is particularly useful in applications where memory, bandwidth, and/or computational power is limited (e.g., a smartcard) and it is in this area that ECC use is expected to grow. A major champion of ECC today is Certicom; readers are urged to see their ECC tutorial.

5.9. The Advanced Encryption Standard (AES) and Rijndael

The search for a replacement to DES started in January 1997 when NIST announced that it was looking for an Advanced Encryption Standard. In September of that year, they put out a formal Call for Algorithms and in August 1998 announced that 15 candidate algorithms were being considered (Round 1). In April 1999, NIST announced that the 15 had been whittled down to five finalists (Round 2): *MARS* (multiplication, addition, rotation and substitution) from IBM; Ronald Rivest's *RC6*; *Rijndael* from a Belgian team; *Serpent*, developed jointly by a team from England, Israel, and Norway; and *Twofish*, developed by Bruce Schneier. In October 2000, NIST announced their selection: Rijndael.

The remarkable thing about this entire process has been the openness as well as the international nature of the "competition." NIST maintained an excellent Web site devoted to keeping the public fully informed, at *http://csrc.nist.gov/archive/aes/*, which is now available as an archive site. Their Overview of the AES Development Effort has full details of the process, algorithms, and comments so I will not repeat everything here.

In October 2000, NIST released the Report on the Development of the Advanced Encryption Standard (AES) that compared the five Round 2 algorithms in a number of categories. The table below summarizes the relative scores of the five schemes (1=low, 3=high):

	Algorithm				
Category	**MARS**	**RC6**	**Rijndael**	**Serpent**	**Twofish**
General security	3	2	2	3	3
Implementation of security	1	1	3	3	2
Software performance	2	2	3	1	1
Smart card performance	1	1	3	3	2
Hardware performance	1	2	3	3	2
Design features	2	1	2	1	3

With the report came the recommendation that Rijndael be named the AES. In February 2001, NIST released the Draft Federal Information Processing Standard (FIPS) AES Specification for public review and comment. AES contains a subset of Rijndael's capabilities (e.g., AES only supports a 128-bit block size) and uses some slightly different nomenclature and terminology, but to understand one is to understand both. The 90-day comment period ended on May 29, 2001 and the U.S. Department of Commerce officially adopted AES in December 2001, published as FIPS PUB 197.

AES (Rijndael) Overview

Rijndael (pronounced as in "rain doll" or "rhine dahl") is a block cipher designed by Joan Daemen and Vincent Rijmen, both cryptographers in Belgium. Rijndael can operate over a variable-length block using variable-length keys; the version 2 specification submitted to NIST describes use of a 128-, 192-, or 256-bit key to encrypt data blocks that are 128, 192, or 256 bits long; note that all nine combinations of key length and block length are possible. The algorithm is written in such a way that block length and/or key length can easily be extended in multiples of 32 bits and it is specifically designed for efficient implementation in hardware or software on a range of processors. The design of Rijndael was strongly influenced by the block cipher called *Square*, also designed by Daemen and Rijmen.

Rijndael is an iterated block cipher, meaning that the initial input block and cipher key undergoes multiple rounds of

transformation before producing the output. Each intermediate cipher result is called a *State*.

For ease of description, the block and cipher key are often represented as an array of columns where each array has 4 rows and each column represents a single byte (8 bits). The number of columns in an array representing the state or cipher key, then, can be calculated as the block or key length divided by 32 (32 bits = 4 bytes). An array representing a State will have **Nb** columns, where **Nb** values of 4, 6, and 8 correspond to a 128-, 192-, and 256-bit block, respectively. Similarly, an array representing a Cipher Key will have **Nk** columns, where **Nk** values of 4, 6, and 8 correspond to a 128-, 192-, and 256-bit key, respectively. An example of a 128-bit State (**Nb**=4) and 192-bit Cipher Key (**Nk**=6) is shown below:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$
$k_{3,0}$	$k_{3,1}$	$k_{3,2}$	$k_{3,3}$	$k_{3,4}$	$k_{3,5}$

The number of transformation rounds (**Nr**) in Rijndael is a function of the block length and key length, and is given by the table below:

No. of Rounds Nr		**Block Size**		
		128 bits Nb = 4	**192 bits Nb = 6**	**256 bits Nb = 8**
Key Size	**128 bits Nk = 4**	10	12	14
	192 bits Nk = 6	12	12	14
	256 bits Nk = 8	14	14	14

Now, having said all of this, the AES version of Rijndael does not support all nine combinations of block and key lengths, but only the subset using a 128-bit block size. NIST calls these supported variants AES-128, AES-192, and AES-256 where the number refers to the key size. The **Nb**, **Nk**, and **Nr** values supported in AES are:

	Parameters		
Variant	**Nb**	**Nk**	**Nr**
AES-128	4	4	10
AES-192	4	6	12
AES-256	4	8	14

The AES/Rijndael cipher itself has three operational stages:

- AddRound Key transformation
- **Nr**-1 Rounds comprising:
  - SubBytes transformation
  - ShiftRows transformation
  - MixColumns transformation
  - AddRoundKey transformation
- A final Round comprising:
  - SubBytes transformation
  - ShiftRows transformation
  - AddRoundKey transformation

The paragraphs below will describe the operations mentioned above. The nomenclature used below is taken from the AES specification although references to the Rijndael specification are made for completeness. The arrays *s* and *s'* refer to the State before and after a transformation, respectively (**NOTE:** The Rijndael specification uses the array nomenclature a and b to refer to the before and after States, respectively). The subscripts *i* and *j* are used to indicate byte locations within the State (or Cipher Key) array.

The SubBytes transformation

The substitute bytes (called *ByteSub* in Rijndael) transformation operates on each of the State bytes independently and changes the byte value. An S-box, or *substitution table*, controls the transformation. The characteristics of the S-box transformation as well as a compliant S-box table are provided in the AES specification; as an example, an input State byte value of 107 (0x6b) will be replaced with a 127 (0x7f) in the output State and an input value of 8 (0x08) would be replaced with a 48 (0x30).

One way to think of the SubBytes transformation is that a given byte in State s is given a new value in State s' according to the S-box. The S-box, then, is a function on a byte in State s so that:

$$s'_{i,j} = \text{S-box}\ (s_{i,j})$$

The more general depiction of this transformation is shown by:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	====>	S-box	====>	$s'_{0,0}$	$s'_{0,1}$	$s'_{0,2}$	$s'_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$				$s'_{1,0}$	$s'_{1,1}$	$s'_{1,2}$	$s'_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$				$s'_{2,0}$	$s'_{2,1}$	$s'_{2,2}$	$s'_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$				$s'_{3,0}$	$s'_{3,1}$	$s'_{3,2}$	$s'_{3,3}$

The ShiftRows transformation

The shift rows (called *ShiftRow* in Rijndael) transformation cyclically shifts the bytes in the bottom three rows of the State array. According to the more general Rijndael specification, rows 2, 3, and 4 are cyclically left-shifted by C1, C2, and C3 bytes, respectively, per the table below:

Nb	C1	C2	C3
4	1	2	3
6	1	2	3
8	1	3	4

The current version of AES, of course, only allows a block size of 128 bits (**Nb** = 4) so that C1=1, C2=2, and C3=3. The diagram below shows the effect of the ShiftRows transformation on State s:

State s					State s'			
$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$	---------- no shift ---------->	$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	----> left-shift by C1 (1) ---->	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$	----> left-shift by C2 (2) ---->	$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$	----> left-shift by C3 (3) ---->	$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

The MixColumns transformation

The mix columns (called *MixColumn* in Rijndael) transformation uses a mathematical function to transform the values of a given column within a State, acting on the four values at one time as if they represented a four-term polynomial. In essence, if you think of MixColumns as a function, this could be written:

$$s'_{i,c} = \text{MixColumns}\ (s_{i,c})$$

for 0<=i<=3 for some column, c. The column position doesn't change, merely the values within the column.

Round Key generation and the AddRoundKey transformation

The AES Cipher Key can be 128, 192, or 256 bits in length. The Cipher Key is used to derive a different key to be applied to the block during each round of the encryption operation. These keys are called the Round Keys and each will be the same length as the block, i.e., **Nb** 32-bit words (words will be denoted W).

The AES specification defines a key schedule by which the original Cipher Key (of length **Nk** 32-bit words) is used to form an *Expanded Key*. The Expanded Key size is equal to the block size times the number of encryption rounds plus 1, which will provide **Nr**+1 different keys. (Note that there are **Nr** encipherment rounds but **Nr**+1 AddRoundKey transformations.)

Consider that AES uses a 128-bit block and either 10, 12, or 14 iterative rounds depending upon key length. With a 128-bit key, for example, we would need 1408 bits of key material (128x11=1408), or an Expanded Key size of 44 32-bit words (44x32=1408). Similarly, a 192-bit key would require 1664 bits of key material (128x13), or 52 32-bit words, while a 256-bit key would require 1920 bits of key material (128x15), or 60 32-bit words. The key expansion mechanism, then, starts with the 128-, 192-, or 256-bit Cipher Key and produces a 1408-, 1664-, or 1920-bit Expanded Key, respectively. The original Cipher Key occupies the first portion of the Expanded Key and is used to produce the remaining new key material.

The result is an Expanded Key that can be thought of and used as 11, 13, or 15 separate keys, each used for one AddRoundKey operation. These, then, are the *Round Keys*. The diagram below shows an example using a 192-bit Cipher Key (**Nk**=6), shown in ***magenta italics***:

Expanded Key:	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	W_{13}	W_{14}	W_{15}	...	W_{44}	W_{45}	W_{46}	W_{47}	W_{48}	W_{49}	W_{50}
Round keys:	Round key 0				Round key 1				Round key 2				Round key 3				...	Round key 11				Round key		

The AddRoundKey (called *Round Key addition* in Rijndael) transformation merely applies each Round Key, in turn, to the State by a simple bit-wise exclusive OR operation. Recall that each Round Key is the same length as the block.

Summary

Ok, I hope that you've enjoyed reading this as much as I've enjoyed writing it — and now let me guide you out of the microdetail! Recall from the beginning of the AES overview that the cipher itself comprises a number of rounds of just a few functions:

- *SubBytes* takes the value of a word within a State and substitutes it with another value by a predefined S-box
- *ShiftRows* circularly shifts each row in the State by some number of predefined bytes
- *MixColumns* takes the value of a 4-word column within the State and changes the four values using a predefined mathematical function
- *AddRoundKey* XORs a key that is the same length as the block, using an Expanded Key derived from the original Cipher Key

```
Cipher (byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])

begin
  byte state[4,Nb]

  state = in

  AddRoundKey(state, w)

  for round = 1 step 1 to Nr-1
    SubBytes(state)
    ShiftRows(state)
    MixColumns(state)
    AddRoundKey(state, w+round*Nb)
  end for

  SubBytes(state)
  ShiftRows(state)
  AddRoundKey(state, w+Nr*Nb)

  out = state
end
```

FIGURE 18: AES pseudocode.

As a last and final demonstration of the operation of AES, Figure 18 is a pseudocode listing for the operation of the AES cipher. In the code:

- *in[]* and *out[]* are 16-byte arrays with the plaintext and cipher text, respectively. (According to the specification, both of these arrays are actually 4***Nb** bytes in length but **Nb**=4 in AES.)
- *state[]* is a 2-dimensional array containing bytes in 4 rows and 4 columns. (According to the specification, this arrays is 4 rows by **Nb** columns.)
- *w[]* is an array containing the key material and is 4*(**Nr**+1) words in length. (Again, according to the specification, the multiplier is actually **Nb**.)
- *AddRoundKey()*, *SubBytes()*, *ShiftRows()*, and *MixColumns()* are functions representing the individual transformations.

5.10. Cisco's Stream Cipher

Stream ciphers take advantage of the fact that:

$$x \text{ XOR } y \text{ XOR } y = x$$

One of the encryption schemes employed by Cisco routers to encrypt passwords is a stream cipher. It uses the

following fixed keystream (thanks also to Jason Fossen for independently extending and confirming this string):

```
dsfd;kfoA,.iyewrkldJKDHSUBsgvca69834ncx
```

When a password is to be encrypted, the password function chooses a number between 0 and 15, and that becomes the offset into the keystream. Password characters are then XORed byte-by-byte with the keystream according to:

$$C_i = P_i \text{ XOR } K_{(offset+i)}$$

where K is the keystream, P is the plaintext password, and C is the ciphertext password.

Consider the following example. Suppose we have the password *abcdefgh*. Converting the ASCII characters yields the hex string 0x6162636465666768.

The keystream characters and hex code that supports an offset from 0 to 15 bytes and a password length up to 24 bytes is:

```
  d s f d ; k f o A , . i y e w r k l d J K D H S U B s g v c a 6 9 8 3 4 n c
                                      x
0x647366643b6b666f412c2e69796577726b6c644a4b444853554273677663613639383334 6e6378
```

Let's say that the function decides upon a keystream offset of 6 bytes. We then start with byte 6 of the keystream (start counting the offset at 0) and XOR with the password:

```
    0x666f412c2e697965
XOR 0x6162636465666768
    ------------------
    0x070D22484B0F1E0D
```

The password would now be displayed in the router configuration as:

```
password 7 06070D22484B0F1E0D
```

where the "7" indicates the encryption type, the leading "06" indicates the offset into the keystream, and the remaining bytes are the encrypted password characters.

(Decryption is pretty trivial so that exercise is left to the reader. If you need some help with byte-wise XORing, see *http://www.garykessler.net/library/byte_logic_table.html*. If you'd like some programs that do this, see *http://www.garykessler.net/software/cisco7.zip*.)

5.11. TrueCrypt

TrueCrypt is an open source, on-the-fly crypto system that can be used on devices supports by Linux, MacOS, and Windows. First released in 2004, TrueCrypt can be employed to encrypt a partition on a disk or an entire disk.

> On May 28, 2014, the TrueCrypt.org Web site was suddenly taken down and redirected to the SourceForge page. Although this paper is intended as a crypto tutorial and not a news source about crypto controversy, the sudden withdrawal of TrueCrypt cannot go without notice. Readers interested in using TrueCrypt should know that the last stable release of the product is v7.1a (February 2012); v7.2, released on May 28, 2014, only decrypts TrueCrypt volumes, ostensibly so that users can migrate to another solution. The current TrueCrypt Web page — TCnext — is TrueCrypt.ch. The TrueCrypt Wikipedia page and accompanying references have some good information about the "end" of TrueCrypt as we knew it.
>
> While there does not appear to be any rush to abandon TrueCrypt at the time of this writing, it is also the case that you don't want to use old, unsupported software for too long. A replacement for TrueCrypt called CipherShed is currently under development. See also "TrueCrypt may live on after all as CipherShed". To date, CipherShed has not produced a product; another — working — fork of TrueCrypt is VeraCrypt.
>
> One final editorial comment. TrueCrypt was ***not*** broken or otherwise compromised! It was withdrawn by its developers for reasons that have not yet been made public but there is no evidence to assume that TrueCrypt has been damaged in any way; on the contrary, two audits, completed in April 2014 and April 2015, found no evidence of backdoors or malicious code. See Steve Gibson's TrueCrypt: Final Release Repository page for more information!

TrueCrypt uses a variety of encryption schemes, including AES, Serpent, and Twofish. A TrueCrypt volume is stored as a file that appears to be filled with random data, thus has no specific file signature. (It is true that a TrueCrypt container will pass a chi-square (X^2) randomness test, but that is merely a general indicator of possibly encrypted content. An additional clue is that a TrueCrypt container will also appear on a disk as a file that is some increment of 512 bytes in size. While these indicators might raise a red flag, they don't rise to the level of clearly

indentifying a TrueCrypt volume.)

When a user creates a TrueCrypt volume, a number of parameters need to be defined, such as the size of the volume and the password. To access the volume, the TrueCrypt program is employed to find the TrueCrypt encrypted file, which is then mounted as a new drive on the host system.

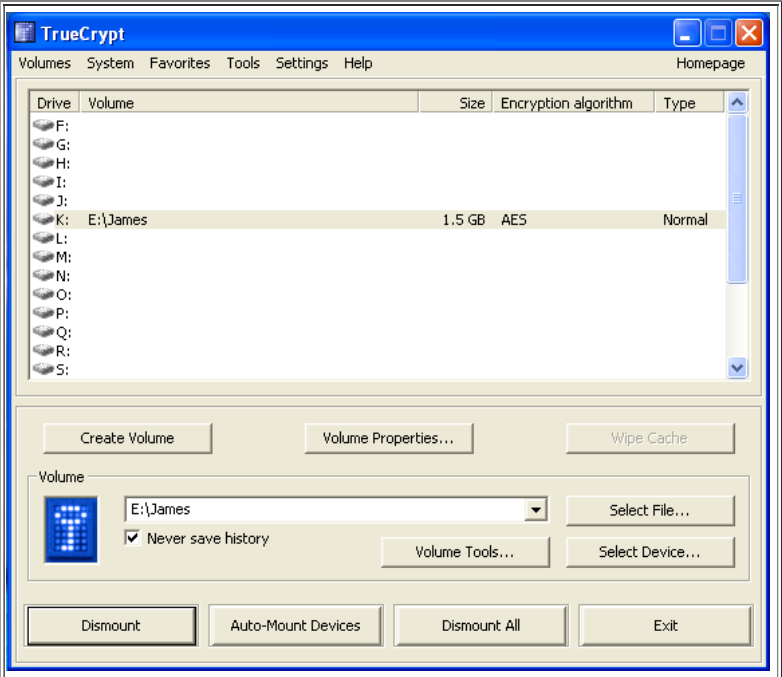


FIGURE 19: TrueCrypt screen shot (Windows).

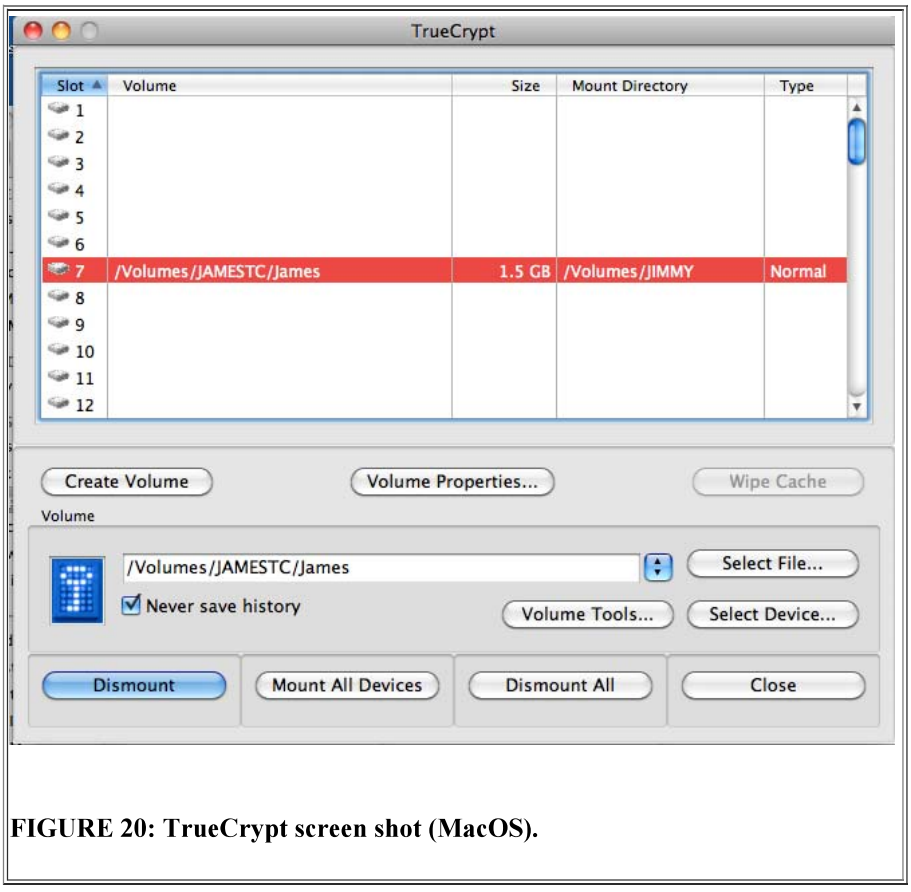


FIGURE 20: TrueCrypt screen shot (MacOS).

Consider this example where an encrypted TrueCrypt volume is stored as a file named *James* on a thumb drive. On a Windows system, this thumb drive has been mounted as device *E:*. If one were to view the *E:* device, any number of

files might be found. The TrueCrypt application is used to mount the TrueCrypt file; in this case, the user has chosen to mount the TrueCrypt volume as device *K:* (Figure 19). Alternatively, the thumb drive could be used with a Mac system, where it has been mounted as the */Volumes/JIMMY* volume. TrueCrypt mounts the encrypted file, *James*, and it is now accessible to the system (Figure 20).

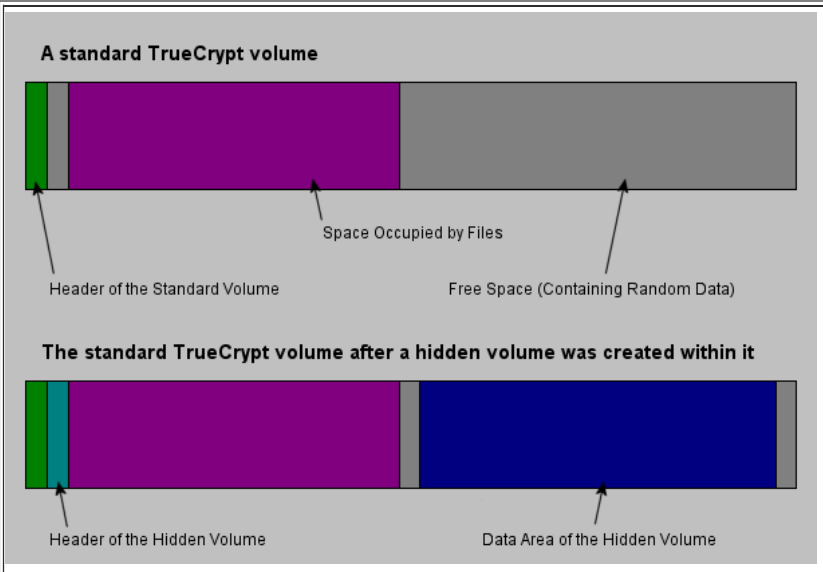


FIGURE 21: TrueCrypt hidden encrypted volume within an encrypted volume (from *http://www.truecrypt.org/images/docs/hidden-volume.gif*).

One of the most interesting — certainly one of the most controversial — features of TrueCrypt is called *plausible deniability*, protection in case a user is "compelled" to turn over the encrypted volume's password. When the user creates a TrueCrypt volume, he/she chooses whether to create a standard or hidden volume. A standard volume has a single password, while a hidden volume is created within a standard volume and uses a second password. As shown in Figure 21, the unallocated (free) space in a TrueCrypt volume is always filled with random data, thus it is impossible to differentiate a hidden encrypted volume from a standard volume's free space.

To access the hidden volume, the file is mounted as shown above and the user enters the hidden volume's password. When under duress, the user would merely enter the password of the standard (i.e., non-hidden) TrueCrypt volume.

More information about TrueCrypt can be found at the TCnext Web Site or in the *TrueCrypt User's Guide (v7.1a)*.

> An active area of research in the digital forensics community is to find methods with which to detect hidden TrueCrypt volumes. Most of the methods do not detect the presence of a hidden volume, per se, but infer the presence by forensic remnants left over. As an example, both Mac and Windows system usually have a file or registry entry somewhere containing a cached list of the names of mounted volumes. This list would, naturally, include the name of TrueCrypt volumes, both standard and hidden. If the user gives a name to the hidden volume, it would appear in such a list. If an investigator were somehow able to determine that there were two TrueCrypt volume names but only one TrueCrypt device, the inference would be that there was a hidden volume. A good summary paper that also describes ways to infer the presence of hidden volumes — at least on some Windows systems — can be found in " Detecting Hidden Encrypted Volumes" (Hargreaves & Chivers).
>
> Having nothing to do with TrueCrypt, but having something to do related to plausible deniability and devious crypto schemes, is a new approach to holding password cracking at bay dubbed *Honey Encryption*. With most of today's crypto systems, decrypting with a wrong key produces digital gibberish while a correct key produces something recognizable, making it easy to know when a correct key has been found. Honey Encryption produces fake data that resembles real data for every key that is attempted, making it significantly harder for an attacker to determine whether they have the correct key or not; thus, if an attacker has a credit card file and tries thousands of keys to crack it, they will obtain thousands of possibly legitimate credit card numbers. See "'Honey Encryption' Will Bamboozle Attackers with Fake Secrets" (Simonite) for some general information or "Honey Encryption: Security Beyond the Brute-Force Bound" (Juels & Ristenpart) for a detailed paper.

5.12. Encrypting File System (EFS)

Microsoft introduced the Encrypting File System (EFS) into the NTFS v3.0 file system. EFS can be used to encrypt individual files, directories, or entire volumes. While off by default, EFS encryption can be easily enabled via Windows Explorer by right-clicking on the file, directory, or volume to be encrypted, selecting Properties, Advanced, and *Encrypt contents to secure data* (Figure 22). Note that encrypted files and directories are displayed in green in Windows Explorer.

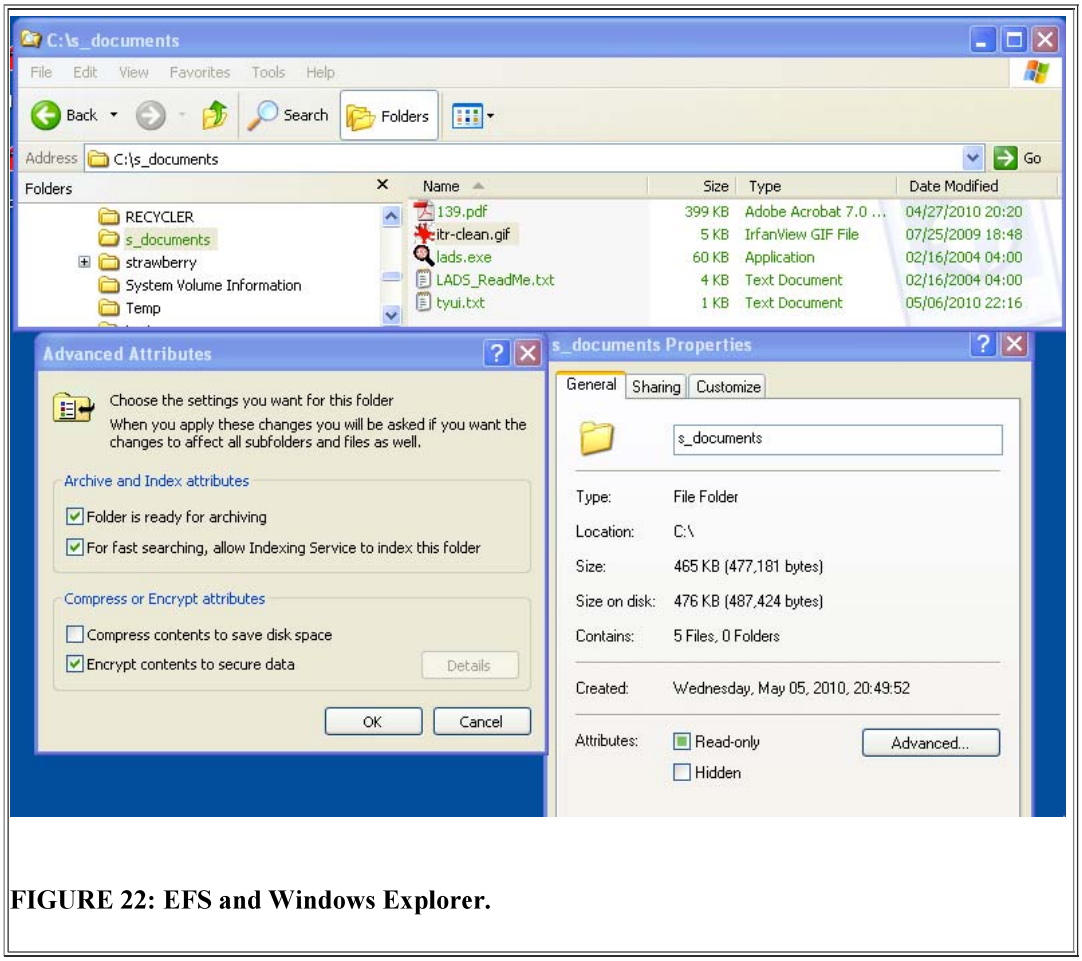


FIGURE 22: EFS and Windows Explorer.

The Windows command prompt provides an easy tool with which to detect EFS-encrypted files on a disk. The `cipher` command has a number of options, but the `/u/n` switches can be used to list all encrypted files on a drive (Figure 23).

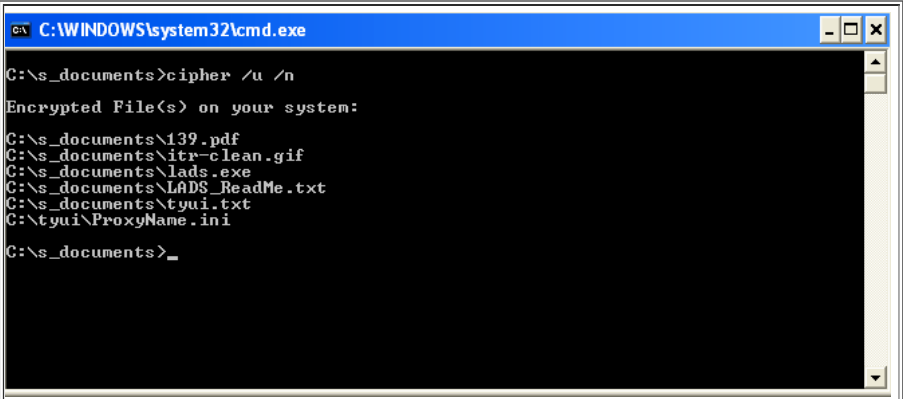


FIGURE 23: The `cipher` command.

EFS supports a variety of secret key encryption schemes, including DES, DESX, and AES, as well as RSA public-key encryption. The operation of EFS — at least at the theoretical level — is clever and simple.

When a file is saved to disk:

- A random File Encryption Key (FEK) is generated by the operating system.
- The file contents are encrypted using one of the SKC schemes and the FEK.
- The FEK is stored with the file, encrypted with the user's RSA public key. In addition, the FEK is encrypted with the RSA public key of any other authorized users and, optionally, a recovery agent's RSA public key.

When the file is opened:

- The FEK is recovered using the RSA private key of the user, another authorized user, or the recovery agent.
- The FEK is used to decrypt the file's contents.

There are weaknesses with the system, most of which are related to key management. As an example, the RSA private key can be stored on an external device such as a floppy disk (yes, really!), thumb drive, or smart card. In practice, however, this is rarely done; the user's private RSA key is on the hard drive. In addition, early EFS implementations (prior to Windows XP SP2) tied the key to the username; later implementations employ the user's password.

A more serious implementation issue is that a backup file named *esf0.tmp* is created prior to a file being encrypted. After the encryption operation, the backup file is deleted — not wiped — leaving an unencrypted version of the file available to be undeleted. For this reason, it is best to use encrypted directories because the temporary backup file is protected by being in an encrypted directory.

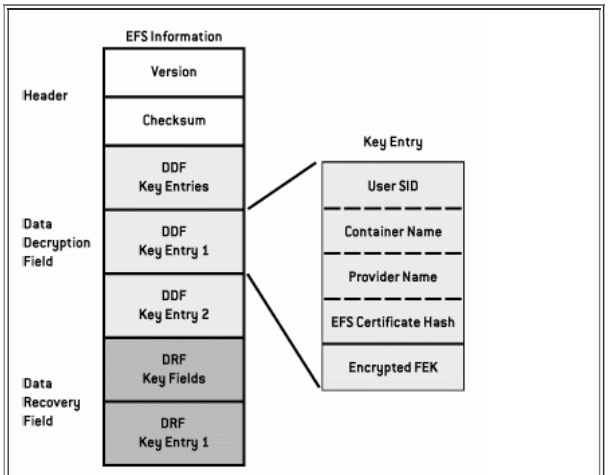


FIGURE 24: EFS key storage. (Source: NTFS.com)

The EFS information is stored as a named stream in the $LOGGED_UTILITY_STREAM Attribute (attribute type 256 [0x100]). This information includes (Figure 24):

- A Data Decryption Field (DDF) for every user authorized to decrypt the file, containing the user's Security Identifier (SID), the FEK encrypted with the user's RSA public key, and other information.
- A Data Recovery Field (DRF) with the encrypted FEK for every method of data recovery

Files in an NTFS file system maintain a number of attributes which contain the system metadata (e.g., the $STANDARD_INFORMATION attribute maintains the file timestamps and the $FILE_NAME attribute contains the file name). Files encrypted with EFS store the keys, as stated above, in a data stream named $EFS within the $LOGGED_UTILITY_STREAM attribute. Figure 25 shows the partial contents of the Master File Table (MFT) attributes for an EFS encrypted file.

```
Master File Table (MFT) Parser V1.4 - Gary C. Kessler (7 June 2012)
   :
   :
0056-0059  Attribute type: 0x10-00-00-00 [$STANDARD_INFORMATION]
0060-0063  Attribute length: 0x60-00-00-00 [96 bytes]
0064       Non-resident flag: 0x00 [Attribute is resident]
   :
   :
0152-0155  Attribute type: 0x30-00-00-00 [$FILE_NAME]
0156-0159  Attribute length: 0x78-00-00-00 [120 bytes]
0160       Non-resident flag: 0x00 [Attribute is resident]
```

```
   :
   :
0392-0395  Attribute type: 0x40-00-00-00 [$VOLUME_VERSION/$OBJECT_ID]
0396-0399  Attribute length: 0x28-00-00-00 [40 bytes]
0400       Non-resident flag: 0x00 [Attribute is resident]
   :
   :
0432-0435  Attribute type: 0x80-00-00-00 [$DATA]
0436-0439  Attribute length: 0x48-00-00-00 [72 bytes]
0440       Non-resident flag: 0x01 [Attribute is non-resident]
   :
   :
0504-0507  Attribute type: 0x00-01-00-00 [$LOGGED_UTILITY_STREAM]
0508-0511  Attribute length: 0x50-00-2E-00 [80 bytes (ignore two high-order bytes)]
0512       Non-resident flag: 0x01 [Attribute is non-resident]
   :
0568-0575  Name: 0x24-00-45-00-46-00-53-00 [$EFS]
```

FIGURE 25: The $LOGGED_UTILITY_STREAM Attribute.

5.13. Some of the Finer Details of RC4

RC4 is a variable key-sized stream cipher developed by Ron Rivest in 1987. RC4 works in output-feedback (OFB) mode, so that the key stream is independent of the plaintext. The algorithm is described in detail in Schneier's "Applied Cryptography," 2/e, pg. 397-398 or the Wikipedia *RC4* article.

RC4 employs an 8x8 substitution box (S-box). The S-box is initialized so that `S[i] = i`, for i=(0,255).

A permutation of the S-box is then performed as a function of the key. The K array is a 256-byte structure that holds the key, repeating itself as necessary so as to be 256 bytes in length (obviously, a longer key results in less repetition). **[[NOTE: All arithmetic below is assumed to be on a byte basis and so is implied to be *modulo 256*.]]**

```
j = 0
for i = 0 to 255
  j = j + S[i] + K[i]
  swap (S[i], S[j])
```

Encryption and decryption are performed by XORing a byte of plaintext/ciphertext with a random byte from the S-box in order to produce the ciphertext/plaintext, as follows:

```
Initialize i and j to zero
```

For each byte of plaintext (or ciphertext):

```
i = i + 1
j = j + S[i]
swap (S[i], S[j])
z = S[i] + S[j]

Decryption: plaintext [i] = S[z] XOR ciphertext [i]
Encryption: ciphertext [i] = S[z] XOR plaintext [i]
```

A Perl implementation of RC4 (fine for academic, but not production, purposes) can be found at *http://www.garykessler.net/software/RC4_v1_3.zip*.

In 2014, Rivest and Schuldt developed a redesign of RC4 called *Spritz*. The main operation of Spritz is similar to the main operation of RC4, except that a new variable, *w*, is added:

```
i = i + w
j = k + S [j + S[i]]
k = i + k + S[j]
swap (S[i], S[j])
z = (S[j + S[i + S[z+k]]]

Decryption: plaintext [i] = S[z] XOR ciphertext [i]
Encryption: ciphertext [i] = S[z] XOR plaintext [i]
```

As seen above, RC4 has two pointers into the S-box, namely, *i* and *j*; Spritz adds a third pointer, *k*.

Pointer *i* move slowly through the S-box; note that it is incremented by 1 in RC4 and by a constant, *w*, in Spritz. Spritz allows *w* to take on any odd value, ensuring that it is always relatively prime to 256. (In essence, RC4 sets *w* to a value of 1.)

In both ciphers, the other pointer(s) — *j* in RC4 or *j* and *k* in Spritz — move pseudorandomly through the S-box. Both ciphers have a single swap of entries in the S-box. Both also produce an output byte, *z*, as a function of the other parameters. Spritz, additionally, includes the previous value of *z* as part of the calculation of the new value of *z*.

6. CONCLUSION... OF SORTS

This paper has briefly described how cryptography works. The reader must beware, however, that there are a number of ways to attack every one of these systems; cryptanalysis and attacks on cryptosystems, however, are well beyond the scope of this paper. In the words of Sherlock Holmes (ok, Arthur Conan Doyle, really), "What one man can invent, another can discover" ("The Adventure of the Dancing Men").

Cryptography is a particularly interesting field because of the amount of work that is, by necessity, done in secret. The irony is that secrecy is *not* the key to the goodness of a cryptographic algorithm. Regardless of the mathematical theory behind an algorithm, the best algorithms are those that are well-known and well-documented because they are also well-tested and well-studied! In fact, *time* is the only true test of good cryptography; any cryptographic scheme that stays in use year after year is most likely a good one. The strength of cryptography lies in the choice (and management) of the keys; longer keys will resist attack better than shorter keys.

The corollary to this is that consumers should run, not walk, away from any product that uses a proprietary cryptography scheme, ostensibly because the algorithm's secrecy is an advantage. The observation that a cryptosystem should be secure even if everything about the system — except the key — is known by your adversary has been a fundamental tenet of cryptography for over 125 years. It was first stated by Dutch linguist Auguste Kerckhoffs von Nieuwenhoff in his 1883 (yes, **18**83) papers titled *La Cryptographie militaire*, and has therefore become known as "Kerckhoffs' Principle."

> *Getting a new crypto scheme accepted, marketed, and, commercially viable is always an interesting challenge. Back in ~2011, for example, a $10,000 challenge page for a new cipher called DioCipher was posted and scheduled to expire on 1 January 2013 — which it did. And that was the last that I heard of DioCipher. I leave it to the reader to consider the validity and usefulness of the public challenge process.*

7. REFERENCES AND FURTHER READING

- Bamford, J. (1983). *The Puzzle Palace: Inside the National Security Agency, America's most secret intelligence organization.* New York: Penguin Books.
- Bamford, J. (2001). *Body of Secrets : Anatomy of the Ultra-Secret National Security Agency from the Cold War Through the Dawn of a New Century.* New York: Doubleday.
- Barr, T.H. (2002). *Invitation to Cryptology.* Upper Saddle River, NJ: Prentice Hall.
- Bauer, F.L. (2002). *Decrypted Secrets: Methods and Maxims of Cryptology,* 2nd ed. New York: Springer Verlag.
- Belfield, R. (2007). *The Six Unsolved Ciphers: Inside the Mysterious Codes That Have Confounded the World's Greatest Cryptographers.* Berkeley, CA: Ulysses Press.
- Denning, D.E. (1982). *Cryptography and Data Security.* Reading, MA: Addison-Wesley.
- Diffie, W., & Landau, S. (1998). *Privacy on the Line.* Boston: MIT Press.
- Electronic Frontier Foundation. (1998). *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design.* Sebastopol, CA: O'Reilly & Associates.
- Federal Information Processing Standards (FIPS) 140-2. (2001, May 25). *Security Requirements for Cryptographic Modules.* Gaithersburg, MD: National Intitute of Standards and Technology (NIST). Retrieved from http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
- Ferguson, N., & Schneier, B. (2003). *Practical Cryptography.* New York: John Wiley & Sons.
- Ferguson, N., Schneier, B., & Kohno, T. (2010). *Cryptography Engineering: Design Principles and Practical Applications.* New York: John Wiley & Sons.
- Flannery, S. with Flannery, D. (2001). *In Code: A Mathematical Journey.* New York: Workman Publishing Company.
- Ford, W., & Baum, M.S. (2001). *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption,* 2nd ed. Englewood Cliffs, NJ: Prentice Hall.
- Garfinkel, S. (1995). *PGP: Pretty Good Privacy.* Sebastopol, CA: O'Reilly & Associates.
- Grant, G.L. (1997). *Understanding Digital Signatures: Establishing Trust over the Internet and Other Networks.* New York: Computing McGraw-Hill.
- Grabbe, J.O. (1997, October 10). Cryptography and Number Theory for Digital Cash. Retrieved from

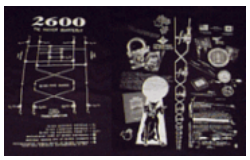
http://www-swiss.ai.mit.edu/6.805/articles/money/cryptnum.htm
- Kahn, D. (1983). *Kahn on Codes: Secrets of the New Cryptology*. New York: Macmillan.
- Kahn, D. (1996). *The Codebreakers: The Story of Secret Writing*, revised ed. New York: Scribner.
- Kaufman, C., Perlman, R., & Speciner, M. (1995). *Network Security: Private Communication in a Public World*. Englewood Cliffs, NJ): Prentice Hall.
- Kessler, G.C. (1999, October). Basics of Cryptography and Applications for Windows NT. *Windows NT Magazine*.
- Kessler, G.C. (2000, February). Roaming PKI. *Information Security Magazine*.
- Kessler, G.C., & Pritsky, N.T. (2000, October). Internet Payment Systems: Status and Update on SSL/TLS, SET, and IOTP. *Information Security Magazine*.
- Koblitz, N. (1994). *A Course in Number Theory and Cryptography,* 2nd ed. New York: Springer-Verlag.
- Levy, S. (1999, April). The Open Secret. *WIRED Magazine*, *7*(4). Retrieved from http://www.wired.com/wired/archive/7.04/crypto.html
- Levy, S. (2001). *Crypto: When the Code Rebels Beat the Government — Saving Privacy in the Digital Age.* New York: Viking Press.
- Mao, W. (2004). *Modern Cryptography: Theory & Practice*. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference.
- Marks, L. (1998). *Between Silk and Cyanide: A Codemaker's War, 1941-1945*. New York: The Free Press (Simon & Schuster).
- Schneier, B. (1996). *Applied Cryptography*, 2nd ed. New York: John Wiley & Sons.
- Schneier, B. (2000). *Secrets & Lies: Digital Security in a Networked World*. New York: John Wiley & Sons.
- Singh, S. (1999). *The Code Book: The Evolution of Secrecy from Mary Queen of Scots to Quantum Cryptography.* New York: Doubleday.
- Smith, L.D. (1943). *Cryptography: The Science of Secret Writing.* New York: Dover Publications.
- Spillman, R.J. (2005). *Classical and Contemporary Cryptology*. Upper Saddle River, NJ: Pearson Prentice-Hall.
- Stallings, W. (2006). *Cryptography and Network Security: Principles and Practice*, 4th ed. Englewood Cliffs, NJ: Prentice Hall.
- Trappe, W., & Washington, L.C. (2006). *Introduction to Cryptography with Coding Theory*, 2nd ed. Upper Saddle River, NJ: Pearson Prentice Hall.
- Young, A., & Yung, M. (2004). *Malicious Cryptography: Exposing Cryptovirology*. New York: John Wiley & Sons.

- On the Web:
  - Bob Lord's Online Crypto Museum
  - Crypto Museum
  - *Crypto-Gram* Newsletter
  - CryptoLog: The Internet Guide to Cryptography (A bit dated...)
  - Cypherpunks -- A history
  - Internet Engineering Task Force (IETF) Security Area
    - An Open Specification for Pretty Good Privacy (openpgp)
    - Common Authentication Technology (cat)
    - IP Security Protocol (ipsec)
    - One Time Password Authentication (otp)
    - Public-Key Infrastructure (X.509) (pkix)
    - S/MIME Mail Security (smime)
    - Simple Public Key Infrastructure (spki)
    - Transport Layer Security (tls)
    - Web Transaction Security (wts)
    - Web Security (websec)
    - XML Digital Signatures (xmldsig)
  - Kerberos: The Network Authentication Protocol (MIT)
  - The MIT Kerberos & Internet trust (MIT-KIT) Consortium (MIT)
  - Peter Gutman's godzilla crypto tutorial
  - Pretty Good Privacy (PGP):
    - The GNU Privacy Guard (GPG)
    - GPGTools
    - The International PGP Home Page
    - The OpenPGP Alliance
  - RSA's Cryptography FAQ (v4.1, 2000)
  - Interspersed in RSA's Public-Key Cryptography Standards (PKCS) pages are a very good set of chapters about cryptography.
  - Ron Rivest's "Cryptography and Security" Page
  - "List of Cryptographers" from U.C. Berkeley

- Software:
  - Wei Dai's Crypto++, a free C++ class library of cryptographic primitives
  - Peter Gutman's cryptlib security toolkit
  - A Perl implementation of RC4 (for academic but not production purposes) can be found at

http://www.garykessler.net/software/RC4_v1_3.zip.
- A Perl program to decode Cisco type 7 passwords can be found at http://www.garykessler.net/software/cisco7_v1.0a.zip.
- Rijndael ANSI C reference code and other implementations

And for a purely enjoyable fiction book that combines cryptography and history, check out Neal Stephenson's *Crytonomicon* (published May 1999). You will also find in it a new secure crypto scheme based upon an ordinary deck of cards (ok, you need the jokers...) called the Solitaire Encryption Algorithm, developed by Bruce Schneier.



Finally, I am not in the clothing business although I do have an impressive t-shirt collection (over 350 and counting!). If you want to proudly wear the DES (well, actually the IDEA) encryption algorithm, try to find *2600 Magazine*'s DES Encryption Shirt, formerly found at http://store.yahoo.com/2600hacker/desenshir.html (left). A t-shirt with Adam Back's RSA Perl code can be found at http://www.cypherspace.org/~adam/uk-shirt.html (right).

APPENDIX. SOME MATH NOTES

A number of readers over time have asked for some rudimentary background on a few of the less well-known mathematical functions mentioned in this paper. Although this is purposely **not** a mathematical treatise, some of the math functions mentioned here are essential to grasping how modern crypto functions work. To that end, some of the mathematical functions mentioned in this paper are defined in greater detail below.

A.1. The Exclusive-OR (XOR) Function

Exclusive OR (XOR) is one of the fundamental mathematical operations used in cryptography (and many other applications). George Boole, a mathematician in the late 1800s, invented a new form of "algebra" that provides the basis for building electronic computers and microprocessor chips. Boole defined a bunch of primitive logical operations where there are one or two inputs and a single output depending upon the operation; the input and output are either TRUE or FALSE. The most elemental Boolean operations are:

- NOT: The output value is the inverse of the input value (i.e., the output is TRUE if the input is false, FALSE if the input is true)
- AND: The output is TRUE if all inputs are true, otherwise FALSE. (E.g., "the sky is blue AND the world is flat" is FALSE while "the sky is blue AND security is a process" is TRUE.)
- OR: The output is TRUE if either or both inputs are true, otherwise FALSE. (E.g., "the sky is blue OR the world is flat" is TRUE and "the sky is blue OR security is a process" is TRUE.)
- XOR (Exclusive OR): The output is TRUE if exactly one of the inputs is TRUE, otherwise FALSE. (E.g., "the sky is blue XOR the world is flat" is TRUE while "the sky is blue XOR security is a process" is FALSE.)

I'll only discuss XOR for now and demonstrate its function by the use of a so-called *truth tables*. In computers, Boolean logic is implemented in *logic gates*; for design purposes, XOR has two inputs (black) and a single output (red), and its logic diagram looks like this:

XOR		Input #1	
		0	**1**
Input #2	**0**	**0**	**1**
	1	**1**	**0**

So, in an XOR operation, the output will be a 1 if one input is a 1; otherwise, the output is 0. The real significance of this is to look at the "identity properties" of XOR. In particular, any value XORed with itself is 0 and any value XORed with 0 is just itself. Why does this matter? Well, if I take my plaintext and XOR it with a key, I get a jumble of bits. If I then take that jumble and XOR it with the same key, I return to the original plaintext.

NOTE: Boolean truth tables usually show the inputs and output as a single bit because they are based on single bit inputs, namely, TRUE and FALSE. In addition, we tend to apply Boolean operations bit-by-bit. For convenience, I have created Boolean logic tables when operating on bytes.

A.2. The *modulo* Function

The *modulo* function is, simply, the remainder function. It is commonly used in programming and is critical to the operation of any mathematical function using digital computers.

To calculate *X modulo Y* (usually written *X mod Y*), you merely determine the remainder after removing all multiples of Y from X. Clearly, the value X mod Y will be in the range from 0 to Y-1.

Some examples should clear up any remaining confusion:

- 15 mod 7 = 1
- 25 mod 5 = 0
- 33 mod 12 = 9
- 203 mod 256 = 203

Modulo arithmetic is useful in crypto because it allows us to set the size of an operation and be sure that we will never get numbers that are too large. This is an important consideration when using digital computers.

A.3. Information Theory and Entropy

Information theory is the formal study of reliable transmission of information in the least amount of space or, in the vernacular of information theory, the fewest *symbols*. For purposes of digital communication, a symbol can be a byte (i.e., an eight-bit octet) or even smaller unit of transmission.

The father of information theory is Bell Labs scientist and MIT professor Claude E. Shannon. His seminal paper, "A Mathematical Theory of Communication" (*The Bell System Technical Journal*, Vol. 27, pp. 379-423, 623-656, July, October, 1948), defined a field that has laid the mathematical foundation for so many things that we take for granted today, from data compression, data storage and communication, and quantum computing to language processing, plagiarism detection and other linguistic analysis, and statistical modeling. And, of course, cryptography — although crypto pre-dates information theory by nearly 2000 years.

There are many everyday computer and communications applications that have been enabled by the formalization of information theory, such as:

- *Lossless data compression*, where the compressed data is an exact replication of the uncompressed source (e.g., PKZip, GIF, PNG, and WAV)
- *Lossy data compression*, where the compressed data can be used to reproduce the original uncompressed source within a certain threshold of accuracy (e.g., JPG and MP3)
- *Coding theory*, which describes the impact of bandwidth and noise on the capacity of data communication channels from modems to Digital Subscriber Line (DSL) services, why a CD or DVD with scratches on the surface can still be read, and error correcting codes used in error-correcting memory chips and forward error correcting satellite communication systems

One of the key concepts of information theory is that of *entropy*. In physics, entropy is a quantification of the disorder in a system; in information theory, in partcular, entropy describes the uncertainty of a random variable, or the randomness of an information symbol. As an example, suppose you have a file and you compress it using PKZip. The two files have the same information content but the smaller (i.e., compressed) file has more entropy because the content is stored in a smaller space (i.e., with fewer symbols) and each data unit has more randomness than in the uncompressed version. In fact, a perfect compression algorithm would result in compressed files with the maximum possible entropy; i.e., the files would contain the same number of 0s and 1s, and they would be distributed within the file in a totally unpredictable, random fashion.

As another example, consider the entropy of passwords (this text is taken from my paper, "Passwords — Strengths And Weaknesses," citing an example in *Firewalls and Internet Security: Repelling the Wily Hacker* by Cheswick & Bellovin [1994]):

> Most Unix systems limit passwords to eight characters in length, or 64 bits. But Unix only uses the seven significant bits of each character as the encryption key, reducing the key size to 56 bits. But even this is not as good as it might appear because the 128 possible combinations of seven bits per character are not equally likely; users usually do not use control characters or non-alphanumeric characters in their passwords. In fact, most users only use lowercase letters in their passwords (and some password systems are case-insensitive, in any case). The bottom line is that ordinary English text of 8 letters has an information content of about 2.3 bits per letter, yielding an 18.4-bit key length for an 8-letter passwords composed of English words. Many people choose names as a password and this yields an even lower information content of about 7.8 bits for the entire 8-letter name. As phrases get longer, each letter only adds about 1.2 to 1.5 bits of information, meaning that a 16-letter password using words from an English phrase only yields a 19- to 24-bit key, not nearly what we might otherwise expect.

Encrypted files tend to have a great deal of randomness. This is why you can encrypt a compressed file but cannot compress an encrypted file; compression algorithms rely on redundancy and repetitive patterns in the source file and such syndromes do not appear in encrypted files.

Randomness is such an integral factor with encrypted files that an entropy test is often the basis for searching for encrypted files. Not all highly randomized files are encrypted, but the more random the contents of a file, the more

likely that the file is encrypted. As an example, the Forensic Toolkit (FTK), software widely used in the computer forensics field, uses the following tests to detect encrypted files:

- *Arithmetic Mean:* Calculated by summing all of the bytes in a file and dividing by the file length; if random, the value should be ~1.75.
- X^2 *Error Percent:* This distribution is calculated for a byte stream in a file; the value indicates how frequently a truly random number would exceed the calculated value.
- *Entropy:* Describes the information density (per Shannon) of a file in bits/character; as entropy approaches 8, there is more randomness.
- *MCPI Error Percent:* The Monte Carlo algorithm uses statistical techniques to approximate the value of π; a high error rate implies more randomness.
- *Serial Correlation Coefficient:* Indicates the amount to which each byte is an e-mail relies on the previous byte. A value close to 0 indicates randomness.

Given this, how do we ensure that crypto algorithms produce random numbers for high levels of entropy? Computers use random number generators (RNGs) for myriad purposes but computers cannot actually generate truly random sequences but, rather, sequences that have mostly random characteristics. To this end, computers use *pseudorandom number generator (PRNG)*, aka *deterministic random number generator*, algorithms. NIST has a series of documents (SP 800-90: Random Bit Generators) that address this very issue:

- *Draft SP 800-90 A Rev. 1: Recommendation for Random Number Generation Using Deterministic Random Bit Generators*
- *Draft SP 800-90 B: Recommendation for the Entropy Sources Used for Random Bit Generation*
- *Draft SP 800-90 C: Recommendation for Random Bit Generator (RBG) Constructions*

SIDEBAR: While the purpose of this document is to be tutorial in nature, I cannot totally ignore the disclosures of Edward Snowden in 2013 about NSA activities related to cryptography. One interesting set of disclosures is around deliberate weaknesses in the NIST PRNG standards at the behest of the NSA. NIST denies any such purposeful flaws but this will be evolving news over time. Interested readers might want to review "NSA encryption backdoor proof of concept published" (M. Lee) or, in particular, "Dual_EC_DRBG backdoor: a proof of concept" (A. Adamantiadis).

For readers interested in learning more about information theory, see the following sites:

- Wikipedia entry for *Information Theory*
- A Short Course in Information Theory (Eight lectures by David J.C. MacKay)
- *Entropy and Information Theory* by Gray (Revised 1st ed., 1991). In 2011, the second edition was published.

Finally, it is important to note that information theory is an continually evolving field. There is an area of research essentially questioning the "power" of entropy in determining the strength of a cryptosystem. An interesting paper about this is "Brute force searching, the typical set and Guesswork" by Christiansen, Duffy, du Pin Calmon, & Médard (2013 IEEE International Symposium on Information Theory); a relatively non-technical overview of that paper can be found at "Encryption Not Backed by Math Anymore" by Hardesty (*DFI News*, 8/15/2013).

ABOUT THE AUTHOR

Gary C. Kessler, Ph.D., CCE, CCFP, CISSP, is the president and janitor of Gary Kessler Associates, an independent consulting and training firm specializing in computer and network security, computer forensics, Internet access issues, and TCP/IP networking. He has written over 60 papers for industry publications, is co-author of *ISDN*, 4th. edition (McGraw-Hill, 1998), and is a past editor-in-chief of the *Journal of Digital Forensics, Security and Law*. Gary is also a Professor of Homeland Security at Embry-Riddle Aeronautical University in Daytona Beach, Florida, where he manages the cybersecrurity minor. He is also a member of the Vermont Internet Crimes Against Children (ICAC) Task Force and North Florida ICAC, and an Adjunct Associate Professor at Edith Cowan University in Perth, Western Australia. Gary was formerly an Associate Professor and Program Director of the M.S. in Information Assurance program at Norwich University in Northfield, Vermont, and he started the M.S. in Digital Investigation Management and undergraduate Computer & Digital Forensics programs at Champlain College in Burlington, Vermont. Gary's e-mail address is *gck@garykessler.net* and his PGP public key can be found at *http://www.garykessler.net/pubkey.html* or on MIT's PGP keyserver (import the latest key!). Some of Gary's other crypto pointers of interest on the Web can be found at his Security-related URLs list.

ACKNOWLEDGEMENTS

An *acknowledgements* section is probably well overdue and so I apologize to all of you who have made helpful comments that remain unacknowledged. If you did make comments that I adopted that improved this paper and I

have failed to recognize you, please remind me!

To get the ball rolling, thanks are offered to Sitaram Chamarty, William R. Godwin, Hugh Macdonald, and Douglas P. McNutt.